PART ONE 设计草稿

实现指令

```
add, sub, ori, lw, sw, beq, lui, jal, jr,nop
```

冒险解决思路

对于我的CPU结构，根据以下四条原则，可以写出所有的冒险处理逻辑。

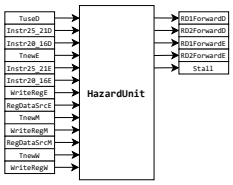
1. 只有D级和E级需要转发。
   只有D级需要延迟。
2. D级需要的冲突数据只会来自于E级，M级，W级（内部转发）。
   E级需要的冲突数据只会来自于M级，W级。
3. 对于不写入寄存器(RegWrite==0)的指令，我们将它的TnewD设为0(不产生)，将它的WriteReg也设为0。

   对于不读取寄存器的指令，我们将它的TuseD设为3(不使用)。
   这样直接使用AT法时就不会出现错误的延迟和转发。
4. 使用AT法
   如果Tnew(E/M/W)==0且寄存器(D/E)==WriteReg(E/M/W)，则需要转发。
   如果TuseD<Tnew(E/M)，则需要延迟。
   其中，根据每一条指令的RegDataSrc，Tnew何时为0是确定的。参见下图或“Control Unit”小节表格。

顶层设计图

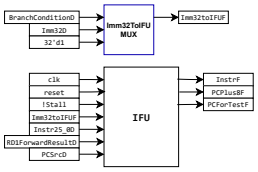
Hazard Unit

Handling Hazards



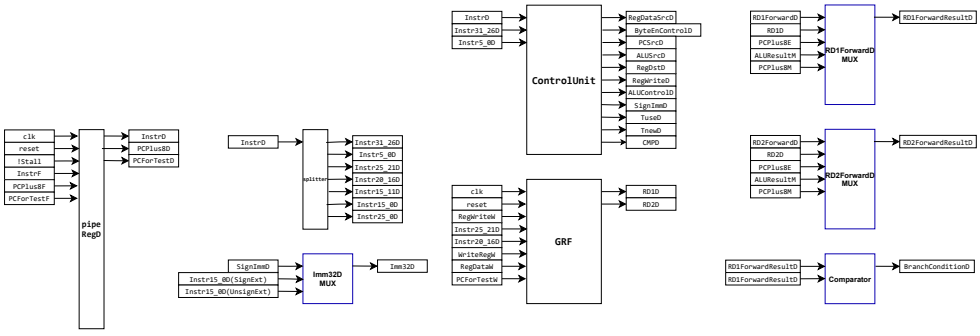
Level F

Instruction Fetch



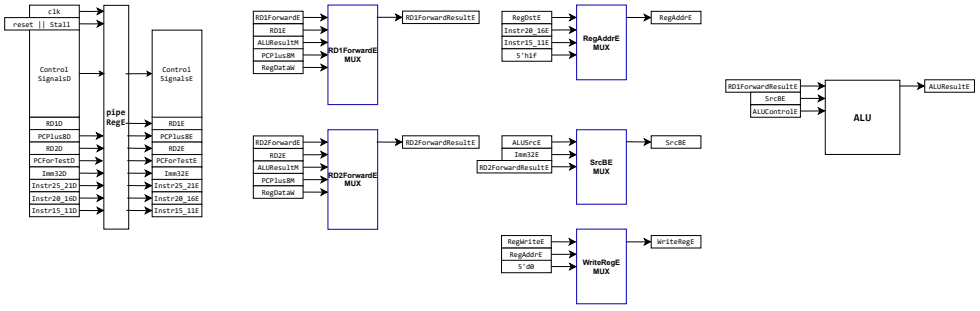
Level D

Instruction Decode



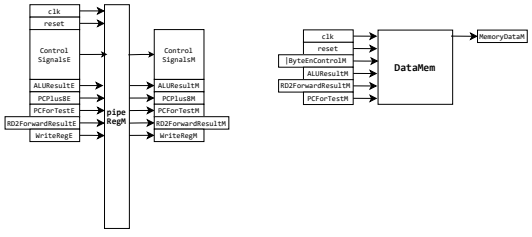
Level E

Excute



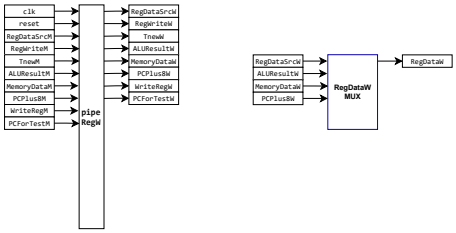
Level M

Memory Access



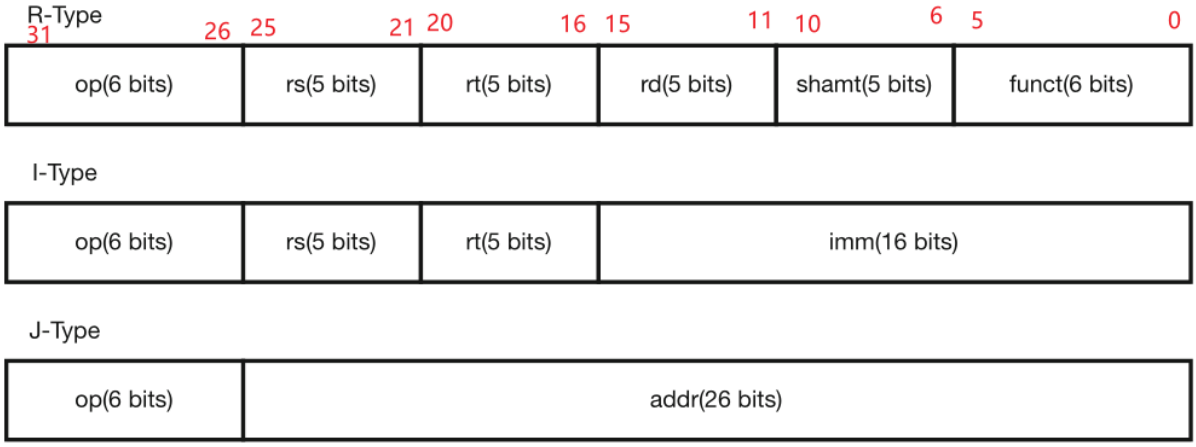
Level W

Register Write Back



各指令机器码

三类指令结构



Opcode & Funct

设计“AND逻辑”时使用：

RRCal Type	Opcode/Funct	RICal Type	Opcode/Funct	LM Type	Opcode/Funct
ADD	000000/100000	ORI	001101	LW	100011
SUB	000000/100010	LUI(AUI)	001111		
SM Type	Opcode/Funct	MD Type	Opcode/Funct	B Type	Opcode/Funct
SW	101011			BEQ	000100
J Type	Opcode/Funct	NOP	0x00000000		
JAL	000011	NOP	0x00000000		
JR	000000/001000				

Control Unit 编码

设计"OR逻辑"时使用：

Hazard Unit

	TnewD	TnewE	TnewM	TnewW
写入ALUResult类	2	1	0	0
写入PCPlus8类	0	0	0	0
写入MemoryData类	3	2	1	0

SignalDecoder_PCsrcDecoder

Instr	PCSrc		Instr	CMP
B Type			B Type	
BEQ	001		BEQ	000
J Type			Others	xxx
JAL	010			
JR	011			
Others	000			

SignalDecoder_ImmEXTDecoder

Instr	SignImm
RICal Type	
ORI	0
LUI(AUI)	1
LM Type	
LW	1
SM Type	
SW	1
B Type	
BEQ	1
Others	0

SignalDecoder_MemDecoder

Instr	ByteEnControl/Meaning		Instr	MemDataControl/Meaning
SM Type			LM Type	
SW	011/1111		LW	011/1111
Others	000/0000		Others	000/0000

SignalDecoder_RegDecoder

Instr	RegWrite*	RegDataSrc*	RegDst*
RRCal Type			
ADD	1	000	001
SUB	1	000	001
RICal Type			
ORI	1	000	000
LUI(AUI)	1	000	000
LM Type			
LW	1	001	000
J Type			
JAL	1	011	010
JR			
Others	0	X(111)	X

SignalDecoder_TimeDecoder

Instr	Tuse	TnewD
RRCal Type		
ADD	1	2
SUB	1	2
RICal Type		
ORI	1	2
LUI(AUI)	1	2
LM Type		
LW	1	3
SM Type		
SW	1	0(不产生)
B Type		
BEQ	0	0(不产生)
J Type		
JAL	0	0(已经产生)
JR	0	0(不产生)
NOP		
NOP	3(不使用)	0(不产生)

SignalDecoder_ALUDecoder

Instr	ALUControl/Meaning	ALUSrc
RRCal Type		
ADD	0000/Add	0
SUB	0001/Sub	0
RICal Type		
ORI	0011/Or	1
LUI(AUI)	0110/Add Upper Imm	1
LM Type		
LW	0000/Add	1
SM Type		
SW	0000/Add	1
Others	X	X

PART TWO 测试方案

工具：使用了石睿知同学的魔改版Mars和Winmerge进行对拍。

数据：使用了https://github.com/refkxh/BUAA_CO_2019Fall的测试数据。

PART THREE 思考题

> 1. 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

答：假如某段程序beq的条件总是不成立，那么就造成了资源的浪费。

> 2. 因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

答：因为延迟槽的存在，PC+4指令必须执行。所以返回的指令需要存入PC+8。

> 3. 我们要求所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），请思考为什么？

答：因为有时间延迟。

> 4. 我们为什么要使用 GPR 内部转发？该如何实现？

答：因为W级向寄存器写入数据，D级从寄存器读出数据，两者同时进行时需要在GPR内部进行转发。

实现代码：

```
assign RD1 = (A1 == 5'd0) ? 32'd0 :
             (WE && A1 == A3) ? WD : registers[A1];
assign RD2 = (A2 == 5'd0) ? 32'd0 :
             (WE && A2 == A3) ? WD : registers[A2];
```

> 5. 我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

答：

需求者	供给者
D级的RD1D，RD2D	E级PCPlus8E，M级ALUResultM, PCPlus8M，W级RegDataW(内部转发)
E级的ALUSrcA, ALUSrcB	M级ALUResultM, PCPlus8M，W级RegDataW

> 6. 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

答：

按照RTL来。

> 7. 简要描述你的译码器架构，并思考该架构的优势以及不足。

答：

采用最普通的集中式译码。优势是硬件省钱，不足是接线很麻烦。