P6 CPU 设计文档

一. 设计要求

支持={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、 SUBU、 MULT、MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}指令的流水线 CPU。

二. 模块规格

1. PC

模块名	功能	输入	输出

PC	上升沿更新 PC;
	EN 为 0 时冻结 PC;
		clk	PC[31:0]

		reset	
		EN	
		NPC[31:0]	

2. NPC

模块名	功能	输入	输出
NPC	根据指令输出正确的 NPC	PC4_D[31:0]	NPC[31:0]
		INSTR[31:0]	
		EXT[31:0]	

3. IM

模块名	功能	输入	输出
IM	根据 PC 输出指令	PC[31:0]	INSTR[31:0]

4. IF/ID

模块名	功能	输入	输出
IF/ID	作为流水线寄存器输出 D 阶段的指令和指令 PC	clk	IR_D[31:0]
		reset	
		EN	PC4_D[31:0]

		INSTR[31:0]	
		PC4[31:0]	

5. GRF

模块名	功能	输入	输出
GRF	Reset 为 1 时置全部寄存器 0； Clk 上升沿时，RD1 读取 GRF[Rs]的值； RD2 读取 GRF[Rt]的值，若 RegWrite 为 1， 则将 WData 的值写入 GRF[Waddr]。	Rs[4:0]	DATA1[31:0] DATA2[31:0]
		Rt[4:0]	
		Waddr[4:0]	
		Wdata[31:0]	
		RegWrite	
		Clk	
		Reset	

6. EXT

模块名	功能	输入	输出
EXT	根据控制信号对指令的十六位立即数进行正确的拓展。	Immediate[15:0]	IME[31:0]
		ExtSrc[1:0]	

7. CMP

模块名	功能	输入	输出
CMP	比较两个输入，equal，greater，g_or_e 为 1 分别表示相等，大于和不小于。	Cmp1[31:0]	Equal
		Cmp2[31:0]	Greater
			G_or_e

8. ID/EX

模块名	功能	输入	输出
ID/EX	E 阶段的流水线寄存器 输出 E 阶段的指令，D 阶段读取的数据及 PC Clk 上升沿更新，reset 或 clr 有效时清空。	IR_D[31:0]	IR_E[31:0]
		PC4_D[31:0]	
		READ1[31:0]	PC4_E[31:0]
		READ2[31:0]	
		EXT[31:0]	RS_E[31:0]
		clk	
		clr	RT_E[31:0]
		reset	EXT_E[31:0]

9. ALU

模块名	功能	输入	输出
ALU	根据 ALUOp 进行各种运算。	A1[31:0]	Result[31:0]
		A2[31:0]	
		ALUOp[3:0]	
		Sa[4:0]	

10. IE/DM

模块名	功能	输入	输出
IE/DM	M 阶段的流水线寄存器 输出 M 阶段的指令，E 阶段的计算结果及 PC Clk 上升沿更新，reset 有效时清空。	Clk	IR_M[31:0]
		Reset	
		IR_E[31:0]	PC4_M[31:0]
		PC4_E[31:0]	AO_M[31:0]
		ALUOUT[31:0]	RT_M[31:0]
		RT_E[31:0]	

11. DM

模块名	功能	输入	输出
DM	时钟上升沿时： MemWrite 为 1 时向 MemAddr 中存储 MemData； Data 输出 MemAddr 中存储的值；	MemAddr[31:0]	Data[31:0]
		MemData[31:0]	
		MemWrite	
		Clk	
		IR_M[31:0]	
		RESET	

12. DM/WR

模块名	功能	输入	输出
D/W	W 阶段的流水线寄存器 输出 W 阶段的指令，M 阶段的读取结果及 PC Clk 上升沿更新，reset 有效时清空。	Clk	IR_W[31:0]
		Reset	
		IR_M[31:0]	AO_W[31:0]
		PC4_M[31:0]	DR_W[31:0]
		AO_M[31:0]	PC4_W[31:0]
		DM[31:0]	

13. MULT_DIV MODULE

模块名	功能	输入	输出
MULT_DIV MODULE	自 Start 信号有效后的第 1 个 clock 上升沿开始,乘除部件开始执行运算,同时 Busy 置位为 1。在运算结果保存到 HI 和 LO 后,Busy 位清除为 0。WE_HI OR LO 为 1 时写入 HI,LO。	D1[31:0]	BUSY
		D2[31:0]	
		CLK	D_HI[31:0]
		RESET	
		WE_HI	
		WE_LO	D_LO[31:0]
		OP[1:0]	
		START	

二. 数据通路设计

根据五级流水线的划分，用四个流水线寄存器 D,E,M,W,构成数据通路的主体部分，考虑到转发，暂停等数据冒险，需要在 D,E,M 阶段加入多路选择器以实现数据冲突的解决，在具体设计上，可列表确定各部件的输入来源，最后汇总以确定数据通路本身是否需要新的多路选择器。

设计表格在附的 EXCEL 文档中。

三. 控制器设计

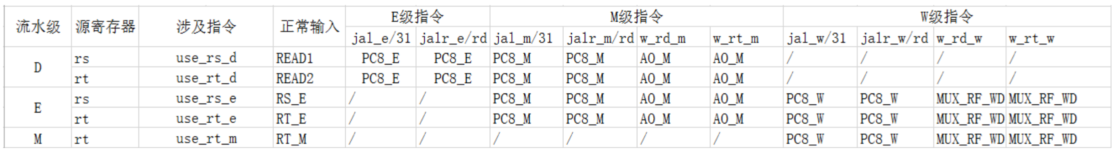
控制器分为三个部分，译码部件控制器，转发控制器，暂停控制器。在设计过程中应该分开考虑。

译码控制器是控制数据通路按照各个指令进行运作的基础部件，其不考虑转发和暂停，假定数据通路以单周期设计为基础，采用分布式译码器，控制器输入为各级流水线寄存器中保存的指令机器码，输出各级流水线所需的控制信号。

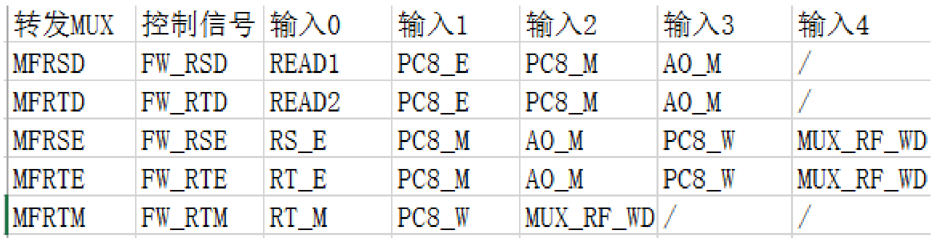
转发控制器的设计可以根据需求供给时间模型来确定其输出的控制信号，其输入为 D,E,MW 阶段的流水线寄存器。首先需要对所有指令以写寄存器和用寄存器的位置进行分类如下：

Use_rs_d: branch,jr,jalr;
Use_rt_d: beq,bne;
Use_rs_e:add,addu,sub,subu,sllv,srlv,srav,and,or,xor,nor,slt,sltu,addi,addiu,andi,ori,xo

ri,slti,sltiu,load,store,mult,multu,div,divu,mthi,mtlo;
Use_rt_e:add,addu,sub,subu,sll,srl,sra,sllv,srlv,srav,and,or,xor,nor,slt,sltu,mult,multu,div,divu,store;
Use_rt_m:store;
Jal_e: jal; Jalr_e: jalr; Jal_m: jal; Jalr_m: jalr; Jal_w: jal; Jalr_w: jalr;
Write_rd_m:add,addu,sub,subu,sll,srl,sra,sllv,srlv,srav,and,or,xor,nor,slt,sltu,mfhi,mflo
Write_rt_m:addi,addiu,andi,ori,xori,lui,slti,sltiu;
Write_rd_w:add,addu,sub,subu,sll,srl,sra,sllv,srlv,srav,and,or,xor,nor,slt,sltu,mfhi,mflo
Write_rt_w:addi,addiu,andi,ori,xori,lui,slti,sltiu,load;



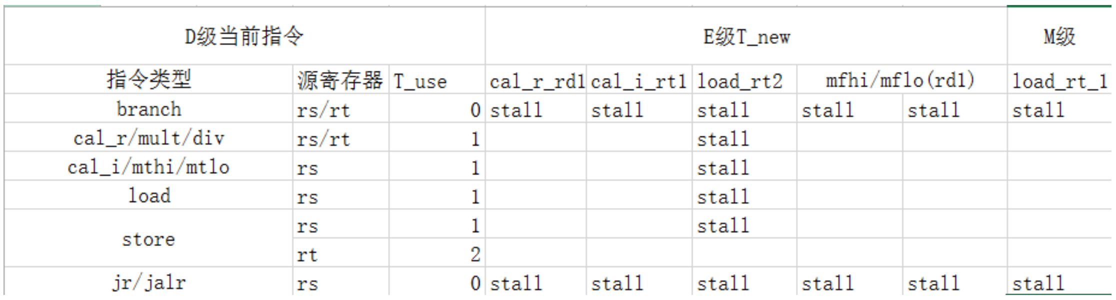
流水级	源寄存器	涉及指令	正常输入	E级指令		M级指令				W级指令			
				jal_e/31	jalr_e/rd	jal_m/31	jalr_m/rd	w_rd_m	w_rt_m	jal_w/31	jalr_w/rd	w_rd_w	w_rt_w
D	rs	use_rs_d	READ1	PC8_E	PC8_E	PC8_M	PC8_M	AO_M	AO_M	/	/	/	/
	rt	use_rt_d	READ2	PC8_E	PC8_E	PC8_M	PC8_M	AO_M	AO_M	/	/	/	/
E	rs	use_rs_e	RS_E	/	/	PC8_M	PC8_M	AO_M	AO_M	PC8_W	PC8_W	MUX_RF_WD	MUX_RF_WD
	rt	use_rt_e	RT_E	/	/	PC8_M	PC8_M	AO_M	AO_M	PC8_W	PC8_W	MUX_RF_WD	MUX_RF_WD
M	rt	use_rt_m	RT_M	/	/	/	/	/	/	PC8_W	PC8_W	MUX_RF_WD	MUX_RF_WD



转发MUX	控制信号	输入0	输入1	输入2	输入3	输入4
MFRSD	FW_RSD	READ1	PC8_E	PC8_M	AO_M	/
MFRTD	FW_RTD	READ2	PC8_E	PC8_M	AO_M	/
MFRSE	FW_RSE	RS_E	PC8_M	AO_M	PC8_W	MUX_RF_WD
MFRTE	FW_RTE	RT_E	PC8_M	AO_M	PC8_W	MUX_RF_WD
MFRTM	FW_RTM	RT_M	PC8_W	MUX_RF_WD	/	/

其中转发 MUX 有 5 个，分别对应 D,E,M 阶段需求的寄存器的值可能在前序指令还未完成寄存器写入时读取，控制单元输出五个控制信号对应之，信号名及其对应如图所示。例如，当 E 阶段是一个 cali 类型的指令，而 M 阶段是一个 calr 类型指令，并且 E 阶段需求的 rs 寄存器刚好是 M 阶段将要更新的寄存器 rd，此时将输出转发控制信号 FW_RSE 为 1，使得 E 阶段将要计算的寄存器值及时更新。

暂停控制器用于在转发无法解决一部分指令组合产生的冲突时，给指令之间加入气泡，机制为锁定 PC 和 D 寄存器，清空 E 寄存器，原来 D 阶段为 A 指令，E 阶段为 B 指令，现在判定 AB 存在无法解决的数据冒险，进行加入气泡操作后，下个时钟上升沿时，指令等价于 D 阶段仍为 A，E 阶段为 nop，M 阶段为 B 指令，于是冲突解决。暂停只需要在 Tuse<Tnew 时进行，否则将降低 CPU 运行效率，列表如下。



D级当前指令			E级T_new					M级
指令类型	源寄存器	T_use	cal_r_rd1	cal_i_rt1	load_rt2	mfhi/mflo(rd1)		load_rt_1
branch	rs/rt	0	stall	stall	stall	stall	stall	stall
cal_r/mult/div	rs/rt	1			stall			
cal_i/mthi/mtlo	rs	1			stall			
load	rs	1			stall			
store	rs	1			stall			
	rt	2						
jr/jalr	rs	0	stall	stall	stall	stall	stall	stall

输入为 D,E,M 级流水线寄存器的指令机器码，输出为 stall，stall 为 1 表示暂停，反之不暂停。表中未列出 busy 信号和 start 信号为 1 时 D 级指令为乘除相关指令时的暂停。

四. 测试代码

首先测试 CPU 能否很好的解决指令的数据冒险，根据所需支持的指令，写成下表。

测试指令类型	冲突类型	代码段
calr	calr_m_rs	addu/subu \$s0, \$s1, \$s2
		addu/subu \$s3, \$s0, \$s1
	calr_m_rt	addu/subu \$s0, \$s1, \$s2
		addu/subu \$s3, \$s1, \$s0
	cali_m_rs	ori \$s0, \$0, 3
		addu/subu \$s1, \$s0, \$0
	cali_m_rt	ori \$s0, \$0, 3
		addu/subu \$s1, \$0, \$s0
	calr_w_rs	addu/subu \$s0, \$s1, \$s2
		lui \$t1, 7
		addu/subu \$s3, \$s0, \$s2
	calr_w_rt	addu/subu \$s0, \$s1, \$s2
		lui \$t1, 7
		addu/subu \$s3, \$s2, \$s0
	cali_w_rs	ori \$s0, \$0, 3
		lui \$t1, 7
		addu/subu \$s1, \$s0, \$s3
	cali_w_rt	ori \$s0, \$0, 3
		lui \$t1, 7
		addu/subu \$s1, \$s3, \$s0
cali	calr_m_rs	addu/subu \$s0, \$s1, \$s2
		ori \$s3, \$s0, 3
	cali_m_rs	ori \$s0, \$0, 3
		ori \$s1, \$s0, 4
	calr_w_rs	addu/subu \$s0, \$s1, \$s2
		lui \$t1, 7
		ori \$s3, \$s0, 3
	cali_w_rs	ori \$s0, \$0, 3
		lui \$t1, 7
		ori \$s3, \$s0, 3

实际测试时根据上表编写了一段尽可能充分的测试代码：

```
.text
init_1:j init_31
    lui $0, 56936
init_2:nop
j init_39
```

```
   addi $19,$0, 7964
init_3:j init_9
   ori $14, 26123
init_4:j init_49
   addiu $30, $0,10873
init_5:j init_32
   lui $21, 53793
init_6:j init_54
   lui $12, 1503
init_7:j init_48
   ori $8, 46619
init_8:j init_61
   lui $11, 40056
init_9:j init_62
   ori $14, 62669
init_10:nop
j init_36
   ori $27, 30718
init_11:nop
j init_58
   ori $15, 30680
init_12:j init_13
   lui $13, 29526
init_13:nop
j init_3
   ori $13, 46993
init_14:j init_19
   ori $4, 49597
init_15:j init_38
   lui $25, 15208
init_16:j init_8
   ori $10, 45682
init_17:j init_16
   addi $10,$0, 47830
init_18:nop
j init_53
   ori $1, 6365
init_19:j init_44
   lui $5, 1722
init_20:j init_2
   lui $19, 49461
init_21:j init_7
   lui $8, 25424
init_22:j init_47
```

```
   lui $6, 55725
init_23:j init_25
   lui $7, 31369
init_24:j init_52
   addiu $31, $0,47120
init_25:nop
j init_21
   ori $7, 36603
init_26:j init_5
   ori $20, 58369
init_27:j init_63
   ori $26, 5967
init_28:nop
j init_4
   ori $29, 52608
init_29:j init_33
   li $3, 27163
init_30:j init_29
   ori $2, 6197
init_31:j init_40
   ori $0, 41556
init_32:nop
j init_45
   ori $21, 58115
init_33:nop
j init_64
   ori $3, 25657
j init_60
   ori $23, 62515
init_35:j init_20
   ori $18, 43690
init_36:j init_56
   lui $28, 12833
init_37:j init_57
   ori $22, 17705
init_38:nop
j init_51
   ori $25, 20578
init_39:j init_26
   lui $20, 11815
init_40:j init_18
   lui $1, 6509
init_41:nop
j init_43
```

```
   ori $17, 25092
init_42:j init_28
   lui $29, 15548
init_43:j init_35
   li $18, 24171
init_44:nop
j init_22
   ori $5, 2433
init_45:j init_37
   addi $22,$0, 62005
init_46:j init_41
   lui $17, 64959
init_47:j init_23
   ori $6, 34267
init_48:j init_50
   lui $9, 37373
init_49:j init_24
   ori $30, 33886
init_50:nop
j init_17
   ori $9, 10605
init_51:j init_27
   lui $26, 3240
init_52:j begin
   lui $31, 36491
init_53:j init_30
   addi $2,$0, 37752
init_54:j init_12
   ori $12, 58450
init_55:j init_15
   li $24, 40393
init_56:j init_42
   ori $28, 29660
init_57:j init_34
   addi $23,$0, 15293
init_58:j init_59
   lui $16, 16008
init_59:j init_46
   ori $16, 27644
init_60:j init_55
   li $24, 48296
init_61:nop
j init_6
   ori $11, 4015
```

```
init_62:j init_11
   lui $15, 45224
init_63:j init_10
   lui $27, 38603
init_64:j init_14
   lui $4, 30386
begin:
# add addi addiu sub subu addu and lui or ori xor
# calr/cali_m_rs
#add
add $8, $9, $10
or $11, $8, $15
add $12, $11, $7
sub $13, $12, $5
add $15, $13, $16
addi $17, $15, 11
add $18, $17, $0
subu $19, $18, $21
add $25, $19, $0
and $20, $25, $14
add $19, $20, $12
or $24, $19, $14
add $15, $24, $1
ori $16, $15, 6
add $4, $16, $6
xor $7, $4, $24
add $3, $7, $11
addiu $15, $3, 11
add $11, $15, $5
#sub
sub $8, $9, $10
or $11, $8, $15
sub $12, $11, $7
sub $15, $13, $16
addi $17, $15, 11
sub $18, $17, $0
subu $19, $18, $21
sub $25, $19, $0
and $20, $25, $14
sub  $19, $20, $12
or $24, $19, $14
sub $15, $24, $1
ori $16, $15, 6
sub $4, $16, $6
```

```
xor $7, $4, $24
sub $3, $7, $11
addiu $15, $3, 11
sub $11, $15, $5
# or
or $11, $8, $15
sub $12, $11, $7
or $15, $13, $16
addi $17, $15, 11
or $18, $17, $0
subu $19, $18, $21
or $25, $19, $0
and $20, $25, $14
or  $19, $20, $12
or $24, $19, $14
or $15, $24, $1
ori $16, $15, 6
or $4, $16, $6
xor $7, $4, $24
or $3, $7, $11
addiu $15, $3, 11
or $11, $15, $5
#xor
addi $17, $15, 11
xor $18, $17, $0
subu $19, $18, $21
xor $25, $19, $0
and $20, $25, $14
xor $19, $20, $12
or $24, $19, $14
xor $15, $24, $1
ori $16, $15, 6
xor $4, $16, $6
xor $7, $4, $24
xor $3, $7, $11
xor $8, $9, $10
or $11, $8, $15
xor $12, $11, $7
sub $13, $12, $5
xor $3, $13, $16
addiu $15, $3, 11
xor $11, $15, $5
#and
and $8, $9, $10
```

```
or $11, $8, $15
and $12, $11, $7
sub $13, $12, $5
and $15, $13, $16
addi $17, $15, 11
and $18, $17, $0
subu $19, $18, $21
and $25, $19, $0
and $20, $25, $14
and $19, $20, $12
or $24, $19, $14
and $15, $24, $1
ori $16, $15, 6
and $4, $16, $6
xor $7, $4, $24
and $3, $7, $11
addiu $15, $3, 11
and $11, $15, $5
#addu
addu $8, $9, $10
or $11, $8, $15
addu $12, $11, $7
sub $13, $12, $5
addu $15, $13, $16
addi $17, $15, 11
addu $18, $17, $0
subu $19, $18, $21
addu $25, $19, $0
and $20, $25, $14
addu $19, $20, $12
or $24, $19, $14
addu $15, $24, $1
ori $16, $15, 6
addu $4, $16, $6
xor $7, $4, $24
addu $3, $7, $11
addiu $15, $3, 11
addu $11, $15, $5
###### calr/cali_m_rt
#######################
#add
add $8, $9, $10
or $11, $4, $8
add $12, $5, $11
```

```
sub $13, $6, $12
add $15, $7, $13
addi $17, $8, 11
add $18, $0, $17
subu $19, $10, $18
add $25, $11, $19
and $20, $12, $25
add $19, $13, $20
or $24, $14, $19
add $15, $15, $24
ori $16, $16, 6
add $4, $7, $16
xor $7, $17, $4
add $3, $15, $7
addiu $15, $17, 11
add $11, $11, $15
#sub
sub $8, $9, $10
or $11, $12, $8
sub $12, $10, $7
sub $15, $13, $12
addi $17, $18, 11
sub $18, $0, $17
subu $19, $14, $18
sub $25, $13, $19
and $20, $9, $25
sub  $19, $14, $20
or $24, $21, $19
sub $15, $20, $24
ori $16, $6, 6
sub $4, $23, $16
xor $7, $14, $4
sub $3, $15, $7
addiu $15, $10, 11
sub $11, $1, $15
# or
or $11, $8, $15
sub $12, $11, $11
or $15, $13, $12
addi $17, $12, 11
or $18, $20, $17
subu $19, $7, $18
or $25, $25, $0
and $20, $20, $25
```

```
or  $19, $11, $20
or $24, $4, $19
or $15, $3, $24
ori $16, $2, 6
or $4, $1, $16
xor $7, $8, $4
or $3, $6, $7
addiu $15, $13, 11
or $11, $15, $15
#xor
xor $11, $8, $15
sub $12, $11, $11
xor $15, $13, $12
addi $17, $12, 11
xor $18, $20, $17
subu $19, $7, $18
xor $25, $25, $0
and $20, $20, $25
xor  $19, $11, $20
or $24, $4, $19
xor $15, $3, $24
ori $16, $2, 6
xor $4, $1, $16
xor $7, $8, $4
xor $3, $6, $7
addiu $15, $13, 11
xor $11, $15, $15
#and
and $11, $8, $15
sub $12, $11, $11
and $15, $13, $12
addi $17, $12, 11
and $18, $20, $17
subu $19, $7, $18
and $25, $25, $0
and $20, $20, $25
and $19, $11, $20
or $24, $4, $19
and $15, $3, $24
ori $16, $2, 6
and $4, $1, $16
xor $7, $8, $4
and $3, $6, $7
addiu $15, $13, 11
```

```
and $11, $15, $15
#addu
addu $8, $9, $10
or $11, $5, $8
addu $12, $1, $11
sub $13, $2, $12
addu $15, $3, $13
addi $17, $4, 11
addu $18, $5, $17
subu $19, $6, $18
addu $25, $7, $19
and $20, $8, $25
addu $19, $9, $20
or $24, $10, $19
addu $15, $11, $24
ori $16, $21, 6
addu $4, $22, $16
xor $7, $5, $4
addu $3, $17, $7
addiu $15, $13, 11
addu $11, $5, $15
# calr/cali_w_rs
#add
add $8, $9, $10
nop
or $11, $8, $15
nop
add $12, $11, $7
nop
sub $13, $12, $5
lui $1, 1213
add $15, $13, $16
or $11, $8, $15
addi $17, $15, 11
lui $1, 1213
add $18, $17, $0
nop
subu $19, $18, $21
lui $1, 1213
add $25, $19, $0
nop
and $20, $25, $14
or $11, $8, $15
add $19, $20, $12
```

```
lui $1, 1213
or $24, $19, $14
lui $1, 1213
add $15, $24, $1
lui $1, 1213
ori $16, $15, 6
nop
add $4, $16, $6
or $11, $8, $15
xor $7, $4, $24
lui $1, 1213
add $3, $7, $11
nop
addiu $15, $3, 11
add $11, $15, $5
#sub
sub $8, $9, $10
or $11, $8, $15
or $11, $8, $15
lui $1, 1213
sub $12, $11, $0
or $11, $8, $15
sub $15, $13, $16
nop
addi $17, $15, 11
lui $1, 1213
sub $18, $17, $0
lui $28, 2015
subu $19, $18, $21
lui $1, 1213
sub $25, $19, $0
nop
and $20, $25, $14
lui $28, 2015
sub  $19, $20, $12
nop
or $24, $19, $14
nop
sub $15, $24, $1
lui $28, 2015
ori $16, $15, 6
lui $28, 2015
sub $4, $16, $6
nop
```

```
xor $7, $4, $24
lui $28, 2015
sub $3, $7, $11
nop
addiu $15, $3, 11
lui $28, 2015
sub $11, $15, $5
# or
or $11, $8, $15
sub $12, $11, $7
lui $28, 2015
or $15, $13, $16
nop
addi $17, $15, 11
lui $28, 2015
or $18, $17, $0
lui $28, 2015
subu $19, $18, $21
lui $28, 2015
or $25, $19, $0
nop
and $20, $25, $14
lui $28, 2015
or  $19, $20, $12
lui $28, 2015
or $24, $19, $14
nop
or $15, $24, $1
nop
ori $16, $15, 6
lui $28, 2015
or $4, $16, $6
lui $28, 2015
xor $7, $4, $24
nop
or $3, $7, $11
lui $28, 2015
addiu $15, $3, 11
lui $28, 2015
or $11, $15, $5
#xor
addi $17, $15, 11
lui $28, 2015
xor $18, $17, $0
```

```
nop
subu $19, $18, $21
lui $28, 2015
xor $25, $19, $0
nop
and $20, $25, $14
lui $28, 2015
xor $19, $20, $12
lui $28, 2015
or $24, $19, $14
lui $28, 2015
xor $15, $24, $1
nop
ori $16, $15, 6
lui $28, 2015
xor $4, $16, $6
lui $28, 2015
xor $7, $4, $24
nop
xor $3, $7, $11
lui $28, 2015
xor $8, $9, $10
nop
or $11, $8, $15
lui $28, 2015
xor $12, $11, $7
nop
sub $13, $12, $5
lui $28, 2015
xor $3, $13, $16
lui $28, 2015
addiu $15, $3, 11
lui $28, 2015
xor $11, $15, $5
#and
and $8, $9, $10
lui $28, 2015
or $11, $8, $15
nop
and $12, $11, $7
lui $28, 2015
sub $13, $12, $5
lui $28, 2015
and $15, $13, $16
```

```
nop
addi $17, $15, 11
lui $28, 2015
and $18, $17, $0
lui $28, 2015
subu $19, $18, $21
lui $28, 2015
and $25, $19, $0
nop
and $20, $25, $14
and $19, $20, $12
nop
or $24, $19, $14
lui $28, 2015
and $15, $24, $1
nop
ori $16, $15, 6
and $15, $24, $1
and $4, $16, $6
and $15, $24, $1
xor $7, $4, $24
and $15, $24, $1
and $3, $7, $11
nop
addiu $15, $3, 11
and $15, $24, $1
and $11, $15, $5
#addu
addu $8, $9, $10
and $15, $24, $1
or $11, $8, $15
and $15, $24, $1
addu $12, $11, $7
nop
sub $13, $12, $5
nop
addu $15, $13, $16
and $15, $24, $1
addi $17, $15, 11
and $15, $24, $1
addu $18, $17, $0
and $15, $24, $1
subu $19, $18, $21
nop
```

```
    addu $25, $19, $0
    and $20, $25, $14
    nop
    addu $19, $20, $12
    and $15, $24, $1
    or $24, $19, $14
    and $15, $24, $1
    addu $15, $24, $1
    and $15, $24, $1
    ori $16, $15, 6
    nop
    addu $4, $16, $6
    and $15, $24, $1
    xor $7, $4, $24
    and $15, $24, $1
    addu $3, $7, $11
    nop
    addiu $15, $3, 11
    and $15, $24, $1
    addu $11, $15, $5
```

实际测试的输出完全符合期望。上面开始的赋值同时也验证了 J 指令的正确性。第三组测试程序作为对新加入的 LW,SW,JAL,JR 的测试，测试了关于 lw，sw 的暂停和转发机制，以及关于 jal 的写入转发。程序如下，注释为期望输出。

```
ori $s1, $0, 2     # 17    2
ori $s2, $0, 3    #  18    3
ori $t2, $0, 4    #  10    4
# calt st
addu $s3, $s1, $s2        # 19   5
sw $s3, 0($0)            # 00000000    5
ori $s3, $0, 9           #  19   9
ori $s4, $0, 7           #  20   7
sw $s3, 0($t2)          #  00000004     9
addu $s3, $s1, $s2       #  19     5
subu $s4, $t2, $t1      #  20     4
sw $s3, 0($s4)         # 00000004   5
subu $s4, $t2, $t1     #  20     4
subu $s3, $s3, $s1     #  19   3
sw $s3, 0($s4)       #  00000004    3
#  lw  sw
lw $s3, 0($0)       #19   5
```

```
sw $s3, 4($0)       #00000004   5
sw $s1, 8($0)      #00000008    2
lw $s3, 8($0)      #19   2
ori $s4, $0, 9     #20   9
sw $s3, 4($0)     #00000004   2
lw $s3, 0($0)     # 19    5
ori $s4, $0, 20   # 20   20
addu $s5, $s4, $s1 # 21   22
sw $s3, 12($t2)   # 00000010   5
# jr
ori $t0, $0, 0x00003070    # 8   00003070
jr $t0
lui $t5, 2      # 13  00020000
lw $s3, 0($0)       #19   5
ori $s1, $0, 0x00003084   # 17 00003084
lui $t4, 3             # 12 00030000
jr $s1
lui $t4, 5             #  12    00050000
lui $t4, 6
ori $s1, $0, 0x000030a4  #  17  000030a4
sw $s1, 0($0)          #  00000000  000030a4
lw $s2, 0($0)          # 18   000030a4
ori $t5, $0, 11       #    13    11
ori $s6, $s2, 0        # 22    000030a4
jr $s2
addu $s7, $s6, $0      #  23  000030a4
lui $s7, 4
jal jump              #  31    000030ac
lui $s7, 9           #   23   00090000
addu $s7, $s7, $s7
jump: addu $s3, $0, $ra    #   19   000030ac
jal jump2               #   31   000030bc
lui $s7, 7              #   23   00070000
addu $s7, $s7, $s7
jump2: lw $s1, 0($0)     #  17   000030a4
ori $s2, $ra, 0        #  18   000030bc
```

截图为运行输出结果，符合期望。对于 sb,sh,lb,lbu,lh,lhu 的测试比较简单，所以不再列举。P6 对于新加的运算类指令，如果分类正确，那么只需要做简单的测试，这里就不列举了。下面贴出关于乘除指令的测试，因为这涉及到一个新的模块。首先测试了正数，负数，正负相乘的结果，然后简单的测试了转发和暂停。

```
li $1, 1
li $2, 2
li $3, 3
li $4, 4
li $5, 5
li $6, 6
li $7, 7
li $8, -8
li $9, -9
li $10, -10
li $11, -11
li $12, -12
li $13, -13
li $20, 20
li $21, 21
li $22, 22
li $23, 23
mult $2, $3
mfhi $16
mflo $17
mult $4, $8
mfhi $18
mflo $17
mult $8, $9
mfhi $16
mflo $17
######
multu $2, $3
mfhi $16
mflo $17
multu $4, $8
mfhi $18
mflo $17
multu $8, $9
mfhi $16
mflo $17
########3
div $5, $2
mfhi $16
mflo $17
div $9, $4
mfhi $18
mflo $17
div $13, $8
```

```
mfhi $16
mflo $17
divu $5, $2
mfhi $16
mflo $17
divu $9, $4
mfhi $18
mflo $17
divu $13, $8
mfhi $16
mflo $17
#####
li $1, 15
li $2, 4
mult $1,$2
mfhi $3
mflo $4
mult $3, $4
mfhi $5
mflo $6
mthi $2
mfhi $10
mtlo $1
mflo $11
mult $1, $2
div $1, $2
mfhi $12
mflo $13
```

五.思考题

1.因为执行乘除法运算需要多个周期的时间，无法在一周期以内计算完，所以不能像普通的加减法一样整合进 ALU 马上计算出结果，同理，因为需要多个时钟周期进行运算，而流水线还在继续执行下去，所以计算结果不可能直接写入寄存器堆，只能先存放在 HI,LO 寄存器中，需要的时候使用 mfhi 和 mflo 指令来读取结果。

2.乘除槽也就是在乘除模块运算时如果之后是一条不需要用到该模块的指令，那么流水线继续执行，如果之后是需要读写 HI,LO 的指令，则暂停流水线。

3.因为 DM 模块本身的延迟就已经很大了，如果数据拓展模块加在 M 级，那么 CPU 的时钟周期将要加长，所以选择在延迟较短的 W 级加入数据拓展模块。

4.在处理字符串时按字节存储更有优势，比如字符串的拼接，比较等操作对于按字节来说很方便，而如果按字节存储则还需要拆成四个字节来进行操作。

5.我的设计风格很传统，按照高老板的 PPT 一步一步搭建，在指令的分类上稍微比 PPT 上的简洁一点，然后最复杂的部分个人认为在于转发和暂停，变量的命名需要简洁而清晰，最终才能够写出完备的控制信号。