5级流水线CPU设计文档+

支持指令

R, add, sub, And, Or, Xor, slt, sltu

addi, andi, xori, ori, lui

lb, lh, lw, sb, sh, sw, lbu, lhu

mult, multu, div, divu, mfhi, mflo, mthi, mtlo

beq, bne, j, jal, jr, bltzal

sll

流程模块设计

CU模块设计

- 相较P4，省去RegWrite信号，直接译出当前指令需要写入的地址，如不需写入，默认写至0，在写入GRF时直接略去
- 直接译出当前指令rs, rt, rd, shamt, imm16, imm26以及所有控制信号供每个阶段选取使用，还需译出Tuse_rs/rt以及E_Tnew与M_Tnew，各级输出对应信号至Conflict模块
- 将指令分类，分为：cal_r,cal_i,md,mt,mf,load,save,branch,branch_ucl,branch_cl,shift,jreg,jadd,jlink（ori被归为cal_i）

```
    assign cal_r=(add||sub||And||Or||Xor||slt||sltu);
    assign cal_i=(addi||andi||xori||ori||lui);

    assign md   =(mult||multu||div||divu);
    assign mf   =(mfhi||mflo);
    assign mt   =(mthi||mtlo);

    assign load=(lw||lh||lhu||lb||lbu);
    assign save=(sw||sh||sb);

    assign branch=(beq||bne||branch_ucl||branch_cl);
    assign branch_ucl=bltzal;
    assign branch_cl=0;

    assign jreg = jr;
    assign jadd = (j||jal);
    assign jlink = jal;

    assign shift=sll;
```

- 控制信号新增：MDU, MDUStart, MDUSelect, MFSelect, ByteSelect, DESelect

- 控制信号调整：GRF_WA, GRF_WDSrc, ALUSelect, **EXTSelect（cal_i各个指令行为不同，注意对照指令集）**,BranchSelect

Port name	Direction	Type	Description
Ins	input	[31:0]	当级指令
branchTrue	input		分支控制信号
控制信号			
GRF_WA	output	[4:0]	写入的地址
GRF_WDSrc	output	[2:0]	写入数据选择
EXTSelect	output		EXT位拓展类型选择
ALUSrc	output		ALU_B的数据源选择
ALUSelect	output	[3:0]	ALU运算类型选择
MDU	output		乘除运算+读写HI LO信号（需要阻塞）
MDUStart	output		乘除运算开始信号
MDUSelect	output	[2:0]	乘除运算+写HI LO功能选择
MFSelect	output	[1:0]	读HI LO功能选择
MemWrite	output		内存写入控制
BranchSelect	output	[3:0]	branch判断类型选择
NPCSelect	output	[2:0]	NPC类型选择
ByteSelect	output	[1:0]	访存数据类型选择
DESelect	output	[2:0]	读取内存后结果拓展类型
指令译码			
opcode	output	[5:0]	
funct	output	[5:0]	
rs	output	[4:0]	
rt	output	[4:0]	
rd	output	[4:0]	
shamt	output	[4:0]	
imm16	output	[15:0]	

Port name	Direction	Type	Description
imm26	output	[25:0]	
T计算			
Tuse_rs	output	[1:0]	
Tuse_rt	output	[1:0]	
E_Tnew	output	[1:0]	
M_Tnew	output	[1:0]	

T计算表格

- 注意新增的乘除指令的AT

Ins	Tuse_rs	Tuse_rt	E_Tnew	M_Tnew
cal_r	1	1	1	
cal_i	1		1	
md	1	1		
mt	1			
mf			1	
load	2		2	1
save	1	2		
branch	0	0		
jreg	0			

Conflit模块设计：AT控制阻塞，直接转发

阻塞：

- D级判断将要使用的寄存器数据是否能得到转发更新，即后续写入相同寄存器的Tnew是否有大于Tuse的，如果有则需要阻塞，以在后续能得到转发更新。特判0号寄存器不需要阻塞，能够直接获得数据0
- 需要得到D级指令rs, rt的Tuse，以及后续E, M级指令的Tnew，在各级CU中计算，发送至冲突单元（W级Tnew全是0不需要考虑，都可以内部转发解决）
- 阻塞时需要暂停更新PC以及F级读出的指令，并且清空D级当前指令的译码输出，以替换为nop空泡
- 新增乘除Stall，在乘除运算即将开始或正在进行时如遇到乘除指令需要Stall

```
reg Stall_rs,Stall_rt,Stall_MDU;        //Stall
always@(*)
begin
    if((D_rs==E_GRF_WA)&&D_rs&&(Tuse_rs<E_Tnew)) Stall_rs=1;
    else if((D_rs==M_GRF_WA)&&D_rs&&(Tuse_rs<M_Tnew)) Stall_rs=1;
    else Stall_rs=0;

    if((D_rt==E_GRF_WA)&&D_rt&&(Tuse_rt<E_Tnew)) Stall_rt=1;
    else if((D_rt==M_GRF_WA)&&D_rt&&(Tuse_rt<M_Tnew)) Stall_rt=1;
    else Stall_rt=0;

    if((MDUBusy||MDUStart)&&MDU) Stall_MDU=1;
    else Stall_MDU=0;
end

assign F_Stall=Stall_rs||Stall_rt||Stall_MDU;
assign D_Stall=Stall_rs||Stall_rt||Stall_MDU;
assign E_Flush=Stall_rs||Stall_rt||Stall_MDU;
```

转发：

- 阻塞后，所有指令在需要读寄存器数据的时候都能够获得后续计算完毕的数据，每级转发出已算出的数据，发送给之前各级即可。
- 需要读寄存器：D级GRF，Branch计算需要rs, rt数据；E级ALU需要rs,rt 数据；M级DM写入数据口需要rt数据
- 需要写寄存器：E级可转发出D级算的PC+8；M级可转发出D级算的PC+8和E级算的的ALU_Y；W级可转发出D级算的PC+8，E级算的的ALU_Y和M级读出的DM数据。**根据当前指令CU译码得到的GRF_WDSrc进行选择。**此外还有W级寄存器写入，可直接内部转发至D级读出

```
assign E_GRF_WD=(GRF_WDSrc==3'b010)?E_PC+8:32'bz;
```

```
assign M_GRF_WD=(GRF_WDSrc==3'b000)?M_ALU_Y:
                (GRF_WDSrc==3'b010)?M_PC+8:
                32'bz;          //M级可转发出D级算
```

```
assign W_GRF_WD=(GRF_WDSrc==3'b000)?W_ALU_Y:
                (GRF_WDSrc==3'b001)?W_DM_RD:
                (GRF_WDSrc==3'b010)?W_PC+8:
                32'bz;                  //W级可
```

```
//内部转发，0号寄存器特判
assign RD1=((A1==WA)&&(A1!=5'b00000))?WD:GRF[A1];
assign RD2=((A2==WA)&&(A2!=5'b00000))?WD:GRF[A2];
```

- 在主模块中，获取各级需要读的寄存器编号（D_rs,D_rt,E_rs,E_rt,M_rt），寄存器原读数（D_rs_data,D_rt_data,E_rs_data,E_rt_data,M_rt_data），写入的寄存器编号（E_GRF_WA,M_GRF_WA,W_GRF_WA）和数据（E_GRF_WD,M_GRF_WD,W_GRF_WD）

- 比较读的编号和写的编号是否有相等的，如有相等的则代表有数据已经更新需要转发，转发优先级为更新次序，最后一次更新优先转发，即优先转发距离需要数据的阶段近的数据，特判如果需要读0号寄存器的数据，直接转发0
- 转发的数据（D_rs_fw,D_rt_fw,E_rs_fw,E_rt_fw,M_rt_fw）发送至各级需要的部分运算，并传递给下一级

```
//Forward
assign D_rs_fw= (D_rs==0)?0:
                (E_GRF_WA==D_rs)?E_GRF_WD:
                (M_GRF_WA==D_rs)?M_GRF_WD:
                D_rs_data;
assign D_rt_fw= (D_rt==0)?0:
                (E_GRF_WA==D_rt)?E_GRF_WD:
                (M_GRF_WA==D_rt)?M_GRF_WD:
                D_rt_data;

assign E_rs_fw= (E_rs==0)?0:
                (M_GRF_WA==E_rs)?M_GRF_WD:
                (W_GRF_WA==E_rs)?W_GRF_WD:
                E_rs_data;
assign E_rt_fw= (E_rt==0)?0:
                (M_GRF_WA==E_rt)?M_GRF_WD:
                (W_GRF_WA==E_rt)?W_GRF_WD:
                E_rt_data;

assign M_rt_fw= (M_rt==0)?0:
                (W_GRF_WA==M_rt)?W_GRF_WD:
                M_rt_data;
```

五级模块设计

- 每个阶段之间以寄存器隔开，寄存器设计在每个模块输出处，使用reg类型
- 每个阶段之间需要流水传递Ins，PC，传给各级CU以译码出当前阶段的rs，rt以及需要写入的地址和写入数据的选择
- 部分阶段前后间需要传递需要使用的NPC, EXTout, ALU_Y, DM_RD

P6更新乘除槽与储存器外置以及按字节访存

- 删去F_IFU与M_DM，添加M_DE与E_MDU
- 乘除槽有两个寄存器，其中数据需要在EMW级流水，以便进行转发，并且需要添加转发信号控制
- 外置储存器需要修改数据通路，前寄存器发送写入数据，后寄存器接收读出数据

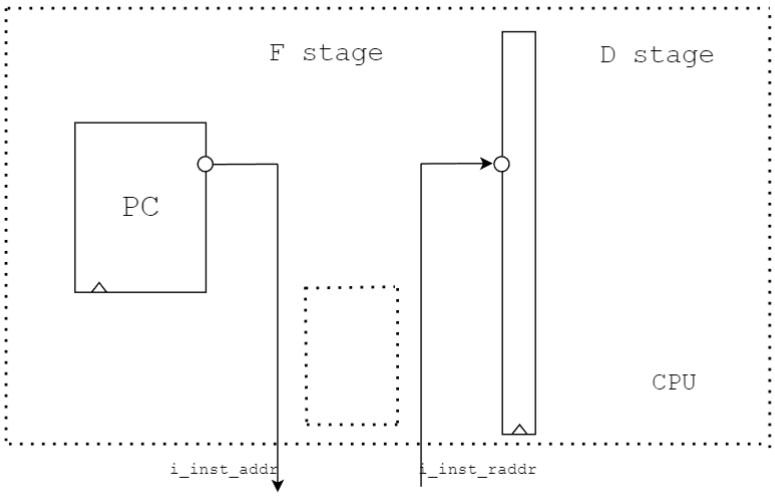
1. Fetch

- 包含FDReg
- Fetch

Port name	Direction	Type	Description

Port name	Direction	Type	Description
clk	input		
reset	input		
F_Flush	input		清空延迟槽信号
F_Stall	input		阻塞更新PC
NPC	input	[31:0]	D级NPC计算出的NPC传入
F_PC	**output**	reg [31:0]	<=NPC，传出至外部指令储存器
F_Ins	**input**	[31:0]	需要从外部指令储存器读入Ins
FD寄存器			
D_Stall	input		阻塞更新FD间寄存器
D_Flush	input		清除延迟槽信号
D_PC	output	reg [31:0]	<=F_PC
D_Ins	output	reg [31:0]	<=F_Ins

- F级与指令储存的数据交换



2. Decode

- 包括D_CU, EXT, NPC (Branch), DEReg

Port name	Direction	Type	Description
clk	input		
reset	input		
D_PC	input	[31:0]	PC流水
D_Ins	input	[31:0]	指令流水

Port name	Direction	Type	Description
Conflict/Forward			
Tuse_rs	output	[1:0]	AT算阻塞
Tuse_rt	output	[1:0]	
D_rs	output	[4:0]	D级指令读寄存器的编号
D_rt	output	[4:0]	
D_rs_data	output	[31:0]	D级指令读寄存器原数据
D_rt_data	output	[31:0]	
D_rs_fw	input	[31:0]	D级转发后寄存器数据
D_rt_fw	input	[31:0]	
EXT			
imm16		[15:0]	EXT输入
EXTSelect			EXT功能选择
D_EXT_out		[31:0]	EXT输出
NPC			
NPCSelect		[2:0]	下一指令地址选择
D_branchTrue			是否分支信号，进入流水
F_PC	input	[31:0]	算NPC用
NPC	output	[31:0]	传给F级IFU
DEReg			
E_Flush	input		阻塞清空DE寄存器
E_PC	output	reg [31:0]	<=D_PC
E_Ins	output	reg [31:0]	<=D_Ins
E_rs_data	output	reg [31:0]	<=**D_rs_fw**
E_rt_data	output	reg [31:0]	<=**D_rt_fw**
E_EXT_out	output	reg [31:0]	<=D_EXT_out
E_branchTrue	output	reg [31:0]	<=D_branchTrue

3. Execute

- 包括E_CU, E_ALU, E_MDU, EMReg
- 需在此处多向Conflict传递MDU指令以及乘除运行信息，并向流水中传递HI, LO以便mf指令W级读取

Port name	Direction	Type	Description
clk	input		
reset	input		
E_PC	input	[31:0]	PC流水
E_Ins	input	[31:0]	指令流水
Conflict/Forward			
E_branchTrue	input		是否分支信号
E_Tnew	output	[1:0]	AT算阻塞
E_rs	output	[4:0]	E级指令读寄存器的编号
E_rt	output	[4:0]	
E_rs_data	output	[31:0]	E级指令读寄存器原数据
E_rt_data	output	[31:0]	
E_GRF_WA	output	[4:0]	E级指令写寄存器的编号
E_rs_fw	input	[31:0]	E级接收转发后寄存器数据
E_rt_fw	input	[31:0]	
GRF_WDSrc		[2:0]	E级指令写寄存器的数据选择
E_GRF_WD	output	[31:0]	E级指令写寄存器的数据
ALU			
E_EXT_out	input	[31:0]	
ALUSrc			ALU_B数据源选择
ALUSelect		[3:0]	ALU功能选择
E_ALU_A		[31:0]	=**E_rs_fw**：ALU_A口数据
E_ALU_B		[31:0]	=**E_rt_fw**/E_EXT_out：ALU_B口数据
MDU			
MDU	output		MDU指令
MDUSelect		[2:0]	CU给MDU的功能选择
MDUStart	output		MDU运算开始

Port name	Direction	Type	Description
MDUBusy	output		MDU运算进行（发给Conflict判断阻塞)
E_HI		[31:0]	待转发的E级MDU的HI结果
E_LO		[31:0]	待转发的E级MDU的LO结果
EMReg			
M_PC	output	reg [31:0]	<=E_PC
M_Ins	output	reg [31:0]	<=E_Ins
M_ALU_Y	output	reg [31:0]	<=E_ALU_Y
M_rt_data	output	reg [31:0]	<**=E_rt_fw**
M_branchTrue	output	reg	<=E_branchTrue
M_HI	output	reg [31:0]	<=E_HI
M_LO	output	reg [31:0]	<=E_LO

- **E_ALU**

Port name	Direction	Type	Description
op	input	[3:0]	
A	input	[31:0]	
B	input	[31:0]	
Y	output	[31:0]	

- **E_MDU**
    - 当指令为mthi, mtlo，将寄存器数据写入HI, LO时，始终上升沿直接给HI, LO赋为A
    - 当为其余四条运算指令时，设置临时计数变量cnt，初始为0，接受到Start信号时，开始设置Busy为1；根据MDU功能选择编码，分别直接计算出HI, LO对应结果赋值，因为其他乘除操作已被阻塞，不会提前读取或写入；设置cnt为5或10，每周期-1，cnt==1代表运算结束，持续保持Busy为5/10周期后将cnt, Busy归零。

Port name	Direction	Type	Description
clk	input		
reset	input		
Start	input		CU传入开始乘除运算信号
MDUSelect	input	[2:0]	CU传入乘除功能选择
A	input	[31:0]	
B	input	[31:0]	

Port name	Direction	Type	Description
Busy	output	reg	正在运算信号
HI	output	reg [31:0]	
LO	output	reg [31:0]	

4. Memory

- 包括M_CU, M_DE
- 因储存器外置，删除DM，加入对字节存取数据的操作，包括通过控制四位ByteEn各位

Port name	Direction	Type	Description
clk	input		
reset	input		
M_PC	input	[31:0]	
M_Ins	input	[31:0]	
Conflict/Forward			
M_branchTrue	input		
M_Tnew	output	[1:0]	AT算阻塞
M_GRF_WA	output	[4:0]	M级指令写寄存器编号
M_GRF_WD	output	[31:0]	M级指令写寄存器数据
M_rt	output	[4:0]	M级指令读寄存器编号
GRF_WDSrc		[2:0]	M级指令写寄存器数据选择
MFSelect		[1:0]	读HI LO功能选择
M_HI	input	[31:0]	待转发的E级MDU的HI结果
M_LO	input	[31:0]	待转发的E级MDU的LO结果
M_ALU_Y	input	[31:0]	待转发的E级ALU计算结果
M_BE（ByteEnable）			
lowAddr		[1:0]	=M_ALU_Y[1:0]，DM写入地址地两位
M_rt_fw	input	[31:0]	M级接收转发后将写入DM的数据
ByteSelect		[1:0]	CU访存数据类型选择
MemWrite			DM写使能
ByteEn	output	reg [3:0]	控制每一位是否读写的信号输出

Port name	Direction	Type	Description
M_DM_WD	output	reg [31:0]	
M_DE（DataExtend）			
DESelect		[2:0]	字节数据拓展类型
M_DM_RDin	input	[31:0]	
M_DM_RDout		[31:0]	
MWReg			
W_PC	output	reg [31:0]	<=M_PC
W_Ins	output	reg [31:0]	<=M_Ins
W_ALU_Y	output	reg [31:0]	<=M_ALU_Y
W_DM_RD	output	reg [31:0]	<=M_DM_RDout
W_branchTrue	output	reg	<=M_branchTrue
W_HI	output	reg [31:0]	<=M_HI
W_LO	output	reg [31:0]	<=M_LO

- **M_BE（计算字节访存使能，调整四字节写入数据）**
    - 合并在Memory中，在写入的条件下，根据写入数据类型和写入地址低两位产生四个字节的每一位控制信号，即四位ByteEn

```
always@(*) begin
    if(MemWrite) begin
        if(ByteSelect==2'b00) begin
            if(lowAddr==0) ByteEn=4'b0001;
            else if(lowAddr==1) ByteEn=4'b0010;
            else if(lowAddr==2) ByteEn=4'b0100;
            else ByteEn=4'b1000;
        end
        else if(ByteSelect==2'b01) begin
            if(lowAddr==0) ByteEn=4'b0011;
            else ByteEn=4'b1100;
        end
        else if(ByteSelect==2'b10) ByteEn=4'b1111;
        else ByteEn=4'b0000;
    end
    else ByteEn=4'b0000;
end
```

    - 后续再根据ByteEn调整将写入内存的数据，需将待写入的字节移动到对应为En1的位置

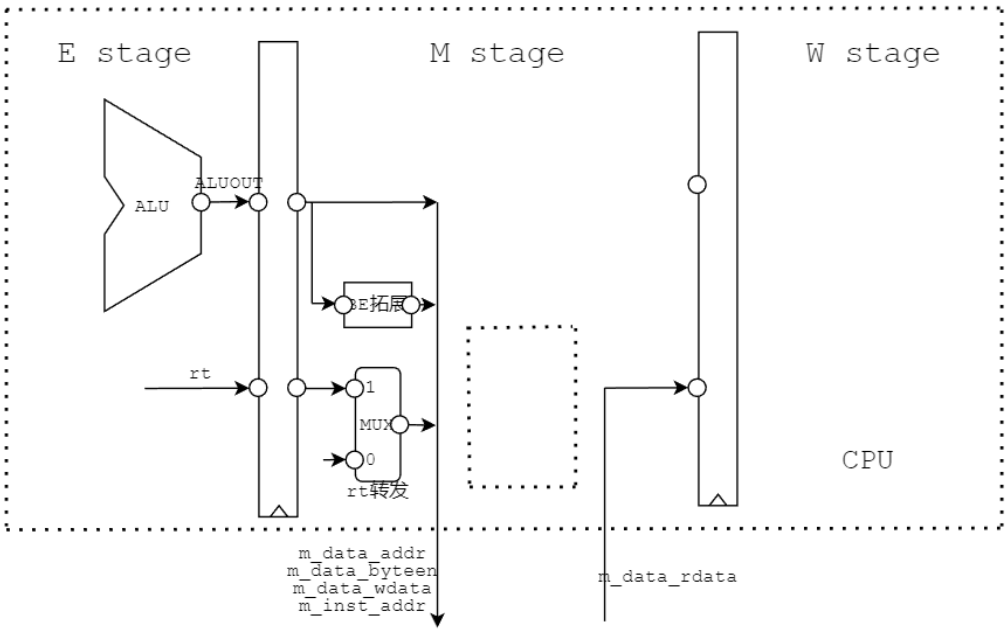
```
always@(*) begin
    if(ByteEn==4'b1111) M_DM_WD=word;
    else if(ByteEn==4'b1100) M_DM_WD={halfword,16'b0};
    else if(ByteEn==4'b0011) M_DM_WD={16'b0,halfword};
    else if(ByteEn==4'b1000) M_DM_WD={byte,24'b0};
    else if(ByteEn==4'b0100) M_DM_WD={8'b0,byte,16'b0};
    else if(ByteEn==4'b0010) M_DM_WD={16'b0,byte,8'b0};
    else if(ByteEn==4'b0001) M_DM_WD={24'b0,byte};
    else M_DM_WD=0;
end
```

- **M_DE（调整内存读出数据，截取需要的字节后拓展）**
    - 注意DESelect种类编码，注意需要将读出字节移动至低位，高位进行拓展补齐

Port name	Direction	Type	Description
lowAddr	input	[1:0]	
DESelect	input	[2:0]	
in	input	[31:0]	
out	output	[31:0]	

```
wire [7:0] byte[3:0];
wire [15:0] halfword[1:0];
assign {byte[3],byte[2],byte[1],byte[0]}=in;     //拆出字节和半字
assign {halfword[1],halfword[0]}=in;

assign out= (DESelect==3'b000)?in:
            (DESelect==3'b001)?{{24{byte[lowAddr][7]}},byte[lowAddr]}:
            (DESelect==3'b010)?{24'b0,byte[lowAddr]}:
            (DESelect==3'b011)?{{16{halfword[lowAddr[1]][15]}},halfword[lowAddr[1]]}:
            (DESelect==3'b100)?{16'b0,halfword[lowAddr[1]]}:
            0;
```

- M级与内存数据交换

- 

5. Writeback

- 包括W_CU

Port name	Direction	Type	Description
clk	input		
reset	input		
W_Ins	input	[31:0]	
W_PC	input	[31:0]	
Conflict/Forward			
W_branchTrue	input		
W_GRF_WA	output	[4:0]	W级指令写寄存器编号
GRF_WDSrc		[2:0]	W级指令写寄存器数据选择
MFSelect		[1:0]	读HI LO功能选择
W_ALU_Y	input	[31:0]	待转发的E级ALU计算结果
W_DM_RD	input	[31:0]	待转发的M级DM读出数据
W_HI	input	[31:0]	待转发的E级MDU的HI结果
W_LO	input	[31:0]	待转发的E级MDU的LO结果
W_GRF_WD	output	[31:0]	W级指令写寄存器数据

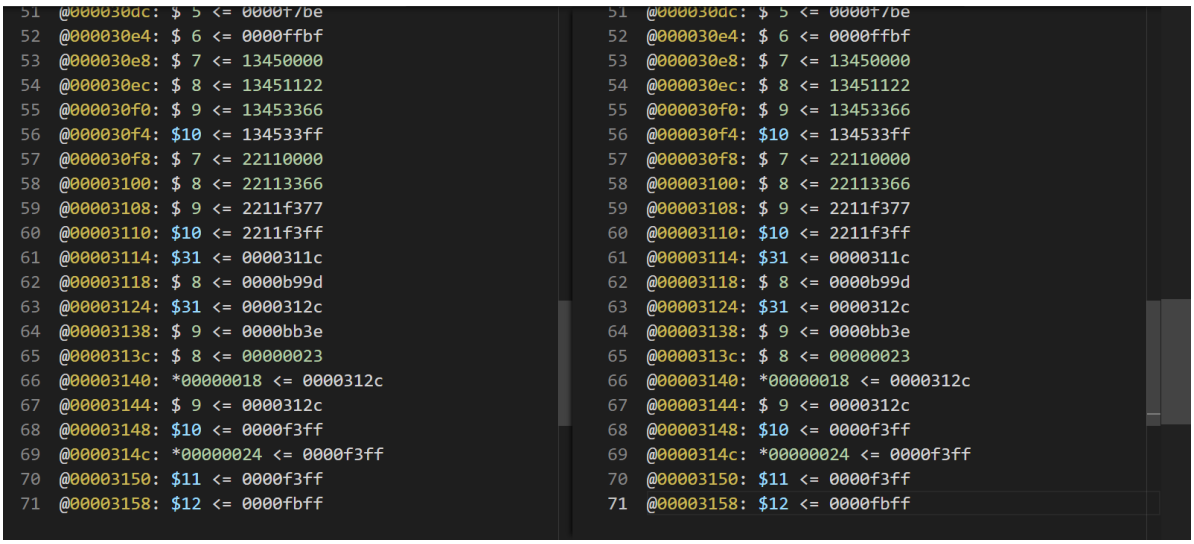
测试(同P5，P6单独构造测试数据)

```
1  .text
2                                       # 13
```

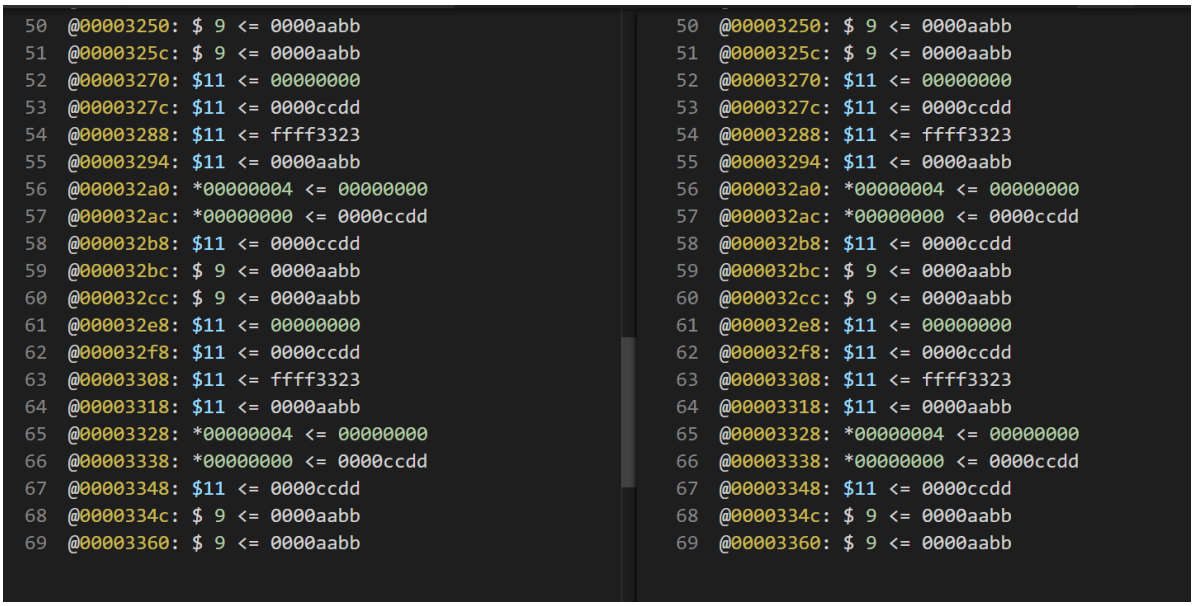
```
    ori     $t0,    $0,         0xadce
    sw      $t0,    12($0)
    ori     $t1,    $t0,        0xdefa
    sw      $t1,    8($0)

    ori     $t3,    $0,         4
    lw      $t2,    8($t3)
    add     $t4,    $t2,        $t0

    ori     $t3,    $0,         15
    lw      $t4,    -7($t3)
    sub     $t5,    $t4,        $t2

# 14
    ori     $t0,    $0,         0xefac
    sw      $t0,    12($0)
    ori     $t1,    $t0,        0xfead
    sw      $t1,    8($0)

    ori     $t3,    $0,         4
    lw      $t2,    8($t3)
    add     $t4,    $t0,        $t2

    ori     $t3,    $0,         21
    lw      $t4,    -13($t3)
    sub     $t5,    $t2,        $t4

# 15
    ori     $t0,    $0,         0x0ace
    sw      $t0,    12($0)
    ori     $t1,    $t0,        0x00a1
    sw      $t1,    8($0)

    ori     $t3,    $0,         4
    lw      $t2,    8($t3)
    nop
    add     $t4,    $t2,        $t0

    ori     $t3,    $0,         15
    lw      $t4,    -7($t3)
    nop
    sub     $t5,    $t4,        $t2

# 16
    lui     $t0,    0x1234
    sw      $t0,    12($0)
    lui     $t1,    0xfead
    sw      $t1,    8($0)

    ori     $t3,    $0,         4
    lw      $t2,    8($t3)
    nop
    add     $t4,    $t0,        $t2

    ori     $t3,    $0,         25
```

```
    lw      $t4,    -17($t3)
    nop
    sub     $t5,    $t2,        $t4

# 17
    lui     $1,     0x13ac
    ori     $2,     0x12ae

    add     $3,     $1,         $2
    ori     $4,     $3,         0xcd12

    sub     $5,     $4,         $1
    ori     $6,     $5,         0x4589

# 18
    lui     $1,     0x56ed
    ori     $2,     0x349a

    add     $3,     $1,         $2
    nop
    ori     $4,     $3,         0xc102

    sub     $5,     $4,         $1
    nop
    ori     $6,     $5,         0x4ea9

# 19
    lui     $7,     0x1345
    ori     $8,     $7,         0x1122

    ori     $9,     $8,         0x3344
    ori     $10,    $9,         0x00ff

# 20
    lui     $7,     0x2211
    nop
    ori     $8,     $7,         0x3366
    nop
    ori     $9,     $8,         0xf111
    nop
    ori     $10,    $9,         0x00ff

# 21
    jal     label1
    ori     $8,     $ra,        0x8899
    nop
    nop
label1:
                                        # 22
    jal     label2
    nop
    nop
    nop
    nop
label2:
```

```
    ori     $9,     $ra,        0xaa12
                                        # 23
    ori     $t0,    $0,         35
    sw      $ra,    24($0)
    lw      $t1,    -11($t0)
    ori     $t2,    $t1,        0xe2df

# 24
    sw      $t2,    36($0)
    lw      $t3,    1($t0)
    nop
    ori     $t4,    $t3,        0xaabb
```



- 对0号寄存器读写测试



注意事项

- 乘除指令的Tuse，Tnew：包括md指令Tuse_rs=Tuse_rt=1; mf指令E_Tnew=1
- 按字节访存的字节位置调整
- 指令功能选择编码
- cal_i指令拓展类型，有有符号，有无符号

思考题

1. 因为硬件运算中乘除法消耗时间很长，需要持续多个周期，放入ALU后会大幅降低频率。独立的HI，LO寄存器方便乘除槽与外部交换数据，并且单独预留给乘除指令，避免其他普通存取导致过多阻塞。
2. 使用移位操作，采用逐位并行的迭代阵列结构，将每个操作数的N位都并行地提交给乘法器

```
module multi_4bits_pipelining(mul_a, mul_b, clk, rst_n, mul_out);
    input [3:0] mul_a, mul_b;
    input clk;
    input rst_n;
    output [15:0] mul_out;

    reg [15:0] mul_out;
    reg [15:0] stored0;
    reg [15:0] stored1;
    reg [15:0] stored2;
    reg [15:0] stored3;
    reg [15:0] stored4;
    reg [15:0] stored5;
    reg [15:0] stored6;
    reg [15:0] stored7;

    reg [15:0] mul_out01;
    reg [15:0] mul_out23;

    reg [15:0] add01;
    reg [15:0] add23;
    reg [15:0] add45;
    reg [15:0] add67;

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            mul_out <= 0;
            stored0 <= 0;
            stored1 <= 0;
            stored2 <= 0;
            stored3 <= 0;
            stored4 <= 0;
                    stored5 <= 0;
            stored6 <= 0;
            stored7 <= 0;

            add01 <= 0;
            add23 <= 0;
            add45 <= 0;
            add67 <= 0;
    end
    else begin
        stored0 <= mul_b[0]? {8'b0, mul_a} : 16'b0;
        stored1 <= mul_b[1]? {7'b0, mul_a, 1'b0} : 16'b0;
        stored2 <= mul_b[2]? {6'b0, mul_a, 2'b0} : 16'b0;
        stored3 <= mul_b[3]? {5'b0, mul_a, 3'b0} : 16'b0;
        stored4 <= mul_b[0]? {4'b0, mul_a, 4'b0} : 16'b0;
        stored5 <= mul_b[1]? {3'b0, mul_a, 5'b0} : 16'b0;
        stored6 <= mul_b[2]? {2'b0, mul_a, 6'b0} : 16'b0;
        stored7 <= mul_b[3]? {1'b0, mul_a, 7'b0} : 16'b0;
```

```
        add01 <= stored1 + stored0;
        add23 <= stored3 + stored2;
        add45 <= stored5 + stored4;
        add67 <= stored7 + stored6;

        mul_out01 <= add01 + add23;
        mul_out23 <= add45 + add67;

        mul_out <= mul_out01 + mul_out23;

    end
    end
endmodule
```

3. 在Start到来的周期开始置1，通过cnt计数周期，计数完毕前始终保持1，完毕后归0
4. 统一所有三种数据类型的访存操作，避免使用过多控制信号。直接用对应字节是否Enable决定是否访存，更加直接清晰。
5. 在lb,sb的情况下是一个字节；lh,sh两个字节；lw,sw四个字节。而按字访存的情况下则始终操作四个字节所以在执行lh,sh,lb,sb指令时按字节读和按字节写的效率会高于按字读和按字写。
6. 将指令分类，译码时不用在每个控制信号与AT计算中添加新指令，便于管理与增量开发，但需要注意某些功能是否相同，具有统一行为，如EXT。采用分布式统一译码，将各级所需控制信号直接独立传递。
7.
    - MDU的指令之间的冲突：在D级检测是否该指令要使用MDU，暂停的条件是要使用MDU并且MDU处于start或busy的状态。

```
ori $t1,5
mthi $t1
mtlo $t1
div $t1,$t2
mfhi $t4
mflo $t5
div $t2,$t3
mfhi $t4
mflo $t5
```

    - MDU与其他指令的冲突：包括md指令Tuse_rs=Tuse_rt=1; mf指令E_Tnew=1

```
here:
ori $t1,5
sw $t1,0($0)
lw $t2,0($0)
div $t1,$t2
mfhi $t3
mflo $t4
beq $t4,$5,here
```

    - 除MDU之外其他指令之间的冲突：指令分类后用指令类型设定对应的Tuse和Tnew，与P5相同

```
ori $t0,5
lui $t1,1
sw $t1,0($0)
lbu $t2,0($0)
or $t3,$t1,$t2
lh $t5,0($0)
slt $t6,$t4,$t5
```

8.
   - 测试单条指令
   - 枚举各指令排列和之间距离
   - 枚举各个寄存器访存，以及HI, LO
   - 枚举各种数据类型和数据的读写