统一定义

- 统一检测时钟上升沿开始运算。
- 使能信号为0时表示真值。包括MemRead，OE，WE，EN等。

PCMux

Input

- PCAdd: PCAdd ＝ PC ＋ imm -- B类指令
- PCRx : PCRx ＝ rx -- JR指令
- PCplus1: PCplus1 = PC + 1 -- PC正常加一
- PCMuxSel: PC选择信号 TYPE SEL_PC IS (PCADD, PCRX, PCPLUS1);

Output

- NewPC:

PC

Input

- NewPC: 新的PC值
- Keep: 是否保持PC而不采用新的PC值

Output

- PC: PC = NewPC

Adder

Input

- PC: PC
- 1

Output

- PCplus1: PCplus1 ＝ PC ＋ 1

InsMemory -- Ram2

Input

- PC: STD_LOGIC_VECTOR(15 downto 0); -- Instruction Address
- Ram2Data: STD_LOGIC_VECTOR(15 downto 0); -- Instruction from Ram2

Output

- Ram2Addr: STD_LOGIC_VECTOR(15 downto 0);
- Ram2CE: Ram2 控制信号
- Ram2OE: Ram2 控制信号
- Ram2WE: Ram2 控制信号
- Instruction: STD_LOGIC_VECTOR(15 downto 0);

MUX_IF_ID

Input

- PCplus1: PC + 1 后的值
- Instruction: IF段获取的指令
- Keep: 是否停止一个周期

Output

- Instruction: STD_LOGIC_VECTOR(15 downto 0)
- PCplus1
- rx: Instruction(10 downto 8)
- ry: Instruction(7 downto 5)
- Imm(11): Instruction(10 downto 0);
- rz: Instruction(4 downto 2)

ControlUnit: Decoder

Input

- Instruction:
- Condition: 来源为data1, BEQZ, BNEZ, BTEQZ, BTNEZ用来比较的数, 根据比较结果来决定 PCMuxSel 的值

Output

- WB 阶段控制信号
  - DstReg: WB阶段的目的寄存器
  - RegWE: WB阶段的写使能
- MEM 阶段控制信号
  - MemRead: 是否读数据, WB阶段的数据选择来源， ALUOut 还是 MemDout 的数据。若读，则为 MemDout, 否则为 ALUOut
  - MemWE: 是否写内存
- EXE 阶段控制信号
  - ALUop: ALU 的操作类型
  - ASrc: ALU 前面的 A 数据选择器选择信号
  - BSrc: ALU 前面的 B 数据选择器选择信号
  - ASrc4: ALU opA 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
  - BSrc4: ALU opB 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
- ID 阶段控制信号
  - ImmeSrc: 包括11位（B指令），8位，5位，4位，3位

    - ZeroExtend: 立即数是否为0扩展
    - Data1Src: 数据1的来源
    - Data2Src: 数据2的来源
    - Read1Register: 数据1
    - Read2Register: 数据2
- IF 阶段控制信号
    - PCMuxSel:

RegisterFile

Input

- PCplus1:向后传递PC，MFPC 指令需要 PC 的值
- Read1Register: rx
- Read2Register: ry
- WriteRegister: WB.DstReg，WB 阶段的目标寄存器
- WriteData: WB.DstVal，WB 阶段要写入目标寄存器的数值
- Data1Src: Data1的来源，TYPE DataSrc IS (None, Rx, Ry, PCplus1, SP, T, IH)
- Data2Src: Data2的来源, DataSrc
- RegWE: WB阶段的寄存器的写使能

> WriteRegister
>
> R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH

> Src:
>
> | **Code** | 意义 |
> | --- | --- |
> | 000 | None |
> | 001 | Rx |
> | 010 | Ry |
> | 011 | PC |
> | 100 | SP |
> | 101 | T |
> | 110 | IH |

Signals

- TYPE Register IS STD_LOGIC_VECTOR(15 downto 0);
- R(0-7);
- SP, T, IH;

Output

> JR 指令的PC需要选择 rx BEQZ, BNEZ 需要比较R[x]的值是否为0或非0，然后给出PCMuxSel信号 BTEQZ, BTNEZ 需要比较T的值。 用 Data1Src 控制信号来统一比较Data1的值。

- Data1: Output数据1
- Data2: Output数据2

MUX_ID_EXE

Input

- data1: RegisterFile 输出的数据 1
- data2: RegisterFile 输出的数据 2
- Immediate: STD_LOGIC_VECTOR(15 downto 0);
- DstReg: 目的寄存器
- RegWE: 寄存器写使能
- MemRead: 内存读, 只有 LW 和 LW_SP 两条指令时为 1,用于 HazardDetectingUnit
- MemWE: 内存写使能
- ALUop: ALU 操作
- ASrc: ALU A 选择器的选择信号
- BSrc: ALU B 选择器的选择信号
- ASrc4: ALU opA 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
- BSrc4: ALU opB 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
- Stall: 气泡，暂停信号

Output

- outData1:
- outData2:
- outImmediate: 立即数
- outDstReg: 目标寄存器: R0 ~ R7, SP, T, IH
- outRegWE: 寄存器写使能
- outMemRead: 内存是否读
- outMemWE: 内存写使能
- outALUop: ALU 操作
- outASrc: ALU A 选择器的选择信号
- outBSrc: ALU B 选择器的选择信号
- outASrc4: 给ForwardingUnit
- outBSrc4: 给ForwardingUnit
- MemWriteData: 要写入内存的寄存器中的值, 用于 SW, SW_SP 指令，如果MemWE,则为Data2，直接传给MUX_EXE_MEM

ImmExtend

author : li

Input

- ImmeSrc: 立即数位数及来源， 包括11位（B指令），8位，5位，4位，3位 下面的常数定义在了 用户Package work.common 中。使用时需要加一句 use work.common.all;

```
000: None
001: 3 bit
010: 4 bit
011: 5 bit
100: 8 bit
101: 11 bit
```

其中3bit的时候为imm(4 downto 2), 符号位是imm(4)。

- ZeroExtend: 是否为0扩展，ZeroExtend为0时采用符号扩展。只有LI指令为ZeroExtend。
- inImme(11): Instruction(10 downto 0);

Output

- Imme(16):

MUX_ALU_A 四选一数据选择器

Input

- data1
- imm
- ExE/MEM.ALUOut
- MEM/WB.DstVal
- ASrc
- ForwardingA: STD_LOGIC_VECTOR(1 downto 0) -- 是否旁路，如果需要旁路，选择 EXE/MEM 的还是MEM/WB

Output

- opA

MUX_ALU_B 四选一数据选择器

Input

- data2
- imm
- EXE/MEM.ALUOut
- MEM/WB.DstVal
- BSrc
- ForwardingB: STD_LOGIC_VECTOR(1 downto 0) -- 是否旁路，如果需要旁路，选择 EXE/MEM 的还是MEM/WB

Output

- opB

ALU

Input

- opA: STD_LOGIC_VECTOR(15 downto 0)
- opB: STD_LOGIC_VECTOR(15 downto 0)
- ALUop:

```
TYPE IS (
    OP_NONE,-- No operation
    OP_ADD, -- F <= A +  B
    OP_SUB, -- F <= A -  B
    OP_AND, -- F <= A &  B
    OP_OR,  -- F <= A |  B
    OP_CMP, -- F <= A != B, not equal
    OP_LT,  -- F <= A <  B
    OP_POS, -- F <= A
    OP_SLL, -- F <= A << B
    OP_SRL, -- F <= A >> B(logical)
    OP_SRA, -- F <= A >> B(arith)
);
```

Output

- F: ALUOut
- T: 标志位, 包括加减法溢出，结果为0等。需要多个标志位

MUX_EXE_MEM

Input

- DstReg: 目标寄存器 R0 ~ R7, SP, T, IH
- RegWE: 寄存器写使能
- MemRead:
- MemWE: 内存写使能
- MemWriteData:
- ALUOut:
- T:
- Stall:

Output

- DstReg: 目标寄存器 R0 ~ R7, SP, T, IH

- RegWE: 寄存器写使能
- MemRead:
- MemWE: 内存写使能
- MemWriteData: 要被写入内存的数据
- ALUOut: ALU 的计算结果, 如果用在MEM段则为地址，否则为WB段的写回寄存器的结果

DM_RAM1 : DataMemory

Input

- CLK: 时钟信号
- RST: reset
- ALUOut: 内存需要读写的地址。需要判断是否是数据区，若是指令内存区则留给Ram2操作，若是串口保留地址则操作串口，若是VGA保留地址则操作VGA_RAM
- MemRead: 是否读内存
- MemWE: 内存写使能
- DstVal: 待写入数据

Output

- Ram1OE: OUT STD_LOGIC;
- Ram1WE: OUT STD_LOGIC;
- Ram1EN: OUT STD_LOGIC;
- Ram1Addr: OUT STD_LOGIC_VECTOR(17 downto 0); -- 需要检测是否为串口地址 0xBF00, 0xBF01等
- Ram1Data: INOUT STD_LOGIC_VECTOR(15 downto 0);
- rdn: OUT STD_LOGIC;
- wrn: OUT STD_LOGIC;
- data_ready: IN STD_LOGIC;
- tbre: IN STD_LOGIC;
- tsre: IN STD_LOGIC;
- vga_wrn: OUT STD_LOGIC;
- vga_data: OUT STD_LOGIC_VECTOR(15 downto 0);
- VGA r,g,b: VGA接口的缓存区
- LedSel: IN STD_LOGIC_VECTOR(15 downto 0);
- LedOut: OUT STD_LOGIC_VECTOR(15 downto 0);
- NumOut: OUT STD_LOGIC_VECTOR(7 downto 0)

MUX_MEM_WB

Input

- ALUOut: ALU 的计算结果
- MemData: Ram1 读出来的数据
- MemRead: 用于选择 ALUOut 和 MemData
- DstReg: 目标寄存器 R0 ~ R7, SP, T, IH
- RegWE: 是否写目标寄存器

Output

- DstReg: WB 阶段的目标寄存器
- RegWE: 是否写目标寄存器
- DestVal: 选择出来的要写入寄存器的值

HazardDetectingUnit

Input

> 检测条件
>
> 1. 上一条指令是 LW 或 LW_SP
> 2. 且它的写入寄存器和当前指令的某一源寄存器相同
>
> ID/EX.MemRead AND (ID/EX.DstReg = IF/ID.rx OR ID/EX.DstReg = IF/ID.ry)

- ID/EXE.MemRead: STD_LOGIC; -- 只有 LW 和 LW_SP 指令时为真）
- ID/EXE.DstReg: STD_LOGIC_VECTOR(3 downto 0);
- ASrc4: STD_LOGIC_VECTOR(2 downto 0); ALU opA 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
- BSrc4: STD_LOGIC_VECTOR(2 downto 0); ALU opB 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH

> 检测条件 2
>
> 1. 上一条指令的目的寄存器是T
> 2. 当前指令的源寄存器是T（BTEQZ, BTNEZ指令）

> 检测条件 (EXE/MEM)
>
> - ALUOut: -- 写入指令的地址
> - MemWE: 写使能

Output

> 让当前指令的控制信号全部为0,即不进行任何写入操作 让PC值保持不变 让IF/ID段寄存器保持不变

- PC_Keep: PC 保持不变
- IFID_Keep: MUX_IF_ID 保持不变
- IDEX_Stall: STD_LOGIC, 暂停信号

ForwardingUnit

Input

> EXE 段检测条件:
>
> - EXE/MEM.RegWE AND EXE/MEM.DstReg != 0 AND
> - EXE/MEM.DstReg = ID/EXE.rx (ry) MEM 段检测条件:
> - MEM/WB.RegWE AND MEM/WB.DstReg != 0 AND
> - (MEM/WB.DstReg = ID/EXE.rx OR MEM/WB.DstReg = ID/EXE.ry)

- exememRegWE: STD_LOGIC;
- exememDstReg: STD_LOGIC_VECTOR(2 downto 0); -- 用于检测 EXE 段数据冲突
- memwbRegWE: STD_LOGIC;
- memwbDstReg: STD_LOGIC_VECTOR(2 downto 0); -- 用于检测 MEM 段数据冲突
- ASrc4: ALU opA 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH
- BSrc4: ALU opB 的确切来源，R0, R1, R2, R3, R4, R5, R6, R7, SP, T, IH

Output

- ForwardingA: 可选的值有 (NotForwarding, EXEForwarding, MEMForwarding)
- ForwardingB: 可选的值有 (NotForwarding, EXEForwarding, MEMForwarding)

结构冲突

地址划分

- 数据:Ram1, 256 * 1024 * 16 bit, 最大地址:0x40000
- 指令:Ram2, 256 * 1024 * 16 bit, 最大地址:0x40000
- 系统程序区:0x0000~0x3FFF,16K
- 用户程序区:0x4000~0x7FFF,16K
- 系统数据区:0x8000~0xBEFF,16K
- 用户数据区:0xC000~0xFFFF,16K
- 串口1数据寄存器:0xBF00
- 串口1状态寄存器:0xBF01
- 串口2数据寄存器:0xBF02
- 串口2状态寄存器:0xBF03

数据冲突 RAW - Read After Write

没有WAW: Write After Write 冲突

没有WAR: Write After Read 冲突

```
ADD R1, R2, R3  ALU->   DM  ->

SUB R4, R1, R5       -> ALU ->

AND R6, R1, R7               -> ALU
```

```
OR  R8, R1, R9

XOR R10,R1, R11
```

ForwardingUnit

1. EXE段数据冲突的检测

当前指令的ID/EXE段和上一条指令的EXE/MEM段

本条指令的源寄存器之一和上一条指令的目的寄存器相同，需要将 rx/ry 保存到ID/EX段 上一条指令需要改写目的寄存器,且不是0寄存器

```
EXE/MEM.RegWE AND EXE/MEM.DstReg != 0 AND
EXE/MEM.DstReg = ID/EXE.rx (ry)
```

2. MEM段数据冲突的检测

```
MEM/WB.RegWE AND MEM/WB.DstReg != 0 AND
(MEM/WB.DstReg = ID/EXE.rx OR MEM/WB.DstReg = ID/EXE.ry)
```

冲突检测单元 HazardDetectingUnit

必须暂停一个周期的指令

```
LW   R2, R1, imm  # R1 = Mem[R2 + sign_extend(imm)]

SUB  R3, R4, R1   # R1 = R4 - R5

AND  R1, R3       # R1 = R1 & R2

OR   R1, R4       # R1 = R1 | R4
```

检测条件:

1. 上一条指令是 LW 或 LW_SP
2. 且它的写入寄存器和当前指令的某一源寄存器相同

```
ID/EX.MemRead AND
```

```
    (ID/EX.DstReg = IF/ID.rx OR ID/EX.DstReg = IF/ID.ry )
```

数据旁路 Forwarding

MEM/WB 寄存器到 EXE/MEM 寄存器后的 ALU 数据选择器

```
(MEM/WB.DstReg = ID/EXE.rx) OR (MEM/WB.DstReg = ID/EXE.ry)
ForwardingA ForwardingB
```

暂停流水线

| 一旦发生此类冲突，暂停流水线一个时钟 | 让当前指令的控制信号全部为0,即不进行任何写入操作 | 让PC值保持不变 让IF/ID段寄存器保持不变 | 将LW指令的结果通过旁路送到ALUInput端

Forwarding逻辑需要增加:

控制冲突

PC add Imm 移到了 ID 段 目前没有准备做分支预测

异常处理

需要实现 EPC 和 CAUSE

EPC

CAUSE