

多周期流水线处理器设计

Mips 50条指令

李厚润 2019202463

实验目标

设计并实现五级流水线MIPS CPU，并且支持一下50 条 MIPS指令

- ALU 指令：ADD / ADDU / SUB / SUBU / SLL / SRL / SRA / SLLV / SRLV / SRAV / AND / OR / XOR / NOR / SLT / SLTU
- 加载立即数到高位：LUI
- 带有立即数的 ALU 指令：ADDI / ADDIU / ANDI / ORI / XORI / SLTI / SLTIU
- 乘除法器指令：MULT / MULTU / DIV / DIVU / MFHI / MTHI / MFLO / MTLO
- 分支指令：BEQ / BNE / BLEZ / BGTZ / BGEZ / BLTZ
- 无条件跳转指令：JR / JALR / J / JAL
- 访存指令：LB / LBU / LH / LHU / LW / SB / SH / SW
- 附加 syscall：执行到 syscall 指令时使用 \$finish 终止仿真

流水线设计

Instruction Fetch

从 Instruction Memory 中取回指令，将其保存到第一级 IF_ID 流水线寄存器中。

Instruction Decode

从第一级 IF_ID 流水线寄存器中取出指令，对其解码分析，并获取一系列 signals，将其全部保存到第二级 ID_EX 流水线寄存器中。

Execution

从第二级 ID_EX 流水线寄存器中取出信号和指令信息，指导 ArithmeticLogicUnit 和 MultiplicationDivisionUnit 运算得到结果，并将其保存到第三级 EX_MEM 流水线寄存器中。

Memory

从第三级 EX_MEM 流水线寄存器中取出地址信息，进行访存操作（如果需要的话），将访存得到的结果保存到第四级 MEM_WB 流水线寄存器中。

Write Back

从第四级 MEM_WB 流水线寄存器中取出之前的所有计算结果和 signals 信号，在信号指导下，将结果保存到目的寄存器中。

signals信号设计

根据第一级 IF_ID 流水线寄存器中保存的指令信息，进行解码分析。比如读取的寄存器 id，ALU 和 DMU 所需要的操作数来源，目的寄存器 id 等等。其中主要信号有：

```
1  typedef struct packed{
2      logic                RegWrite;    //result whether needs to be written
      into register
3      read_type_t          readType;    // read type such as bytesigned,
      Half_word_signed, for lw, lb...
4      logic                MemWrite;    // result whether needs to be written
      into data memory
5      write_type_t          writeType;   // read type such as bytes, Half_word,
      for sb, sw...
6      alu_operation_t       ALUOPCode;  //alu operation code
7      md_u_operation_t       DMUOPCode; //dmu operation code
8      logic                md_u_start;  //dmu start working sign
9      memery_to_reg_t       MemtoReg;   // the data type to be written into
      register
10     reg_id_t              RegDst;      // register destination
11     reg_id_t              Read1ID;
12     reg_id_t              Read2ID;
13     ALU_DMU_source_t       ALUDMUOperand1;
14     ALU_DMU_source_t       ALUDMUOperand2;
15     jump_condition_t       jmpCondition; //jump condition, such as beq, bne
      and so on
16     jump_dest_source_t     jmpDest;    //the source of jump addr, such as
      from register or absolute addr...
17 }control_signals_t;
```

数据冒险处理设计

解决办法

数据旁路转发

- 冲突原因：流水线中的某个阶段依赖还未写入寄存器中的某个值。
- 解决办法：被需要的某个寄存器的值并不需要彻底走完五级流水线才能获取，我们可以通过提前数据转发到需要的地方即可。
- 数据冒险举例以及解决办法：

```
1  case1:
2  add $1, $2, $3
3  ori $1, $1, 100
4  //将add 的值(应该存入$1)从EX_MEM流水线寄存器转发到EX stage, ori便不需要阻塞。
```

阻塞与气泡

- 冲突原因：流水线中的某个阶段需要还未写入寄存器中的某个值，并且数据旁路转发也无法解决。或者由于结构冒险(MDU在忙)。
- 解决办法：假设需要被阻塞的 stage 为 A，下一 stage 为 B，那么我们冻结 A 与 B 之间以及之前的所有流水线寄存器和 PC 指针，在 B 阶段插入气泡。其中只可能在 Decode、Exe 阶段插入气泡，Memory 和 WriteBack 阶段不可能阻塞。
- 数据冒险举例以及解决办法：

```
1  Case1
```

```

2  lw $1, 0($2)
3  ori $1, $1, 100
4  //阻塞一个周期
5
6  Case2
7  add $1, $2, $3
8  beq $1, $zero
9  // 阻塞一个周期
10
11 Case3
12 lw $1, 0($2)
13 beq $1, $zero
14 // 阻塞两个周期

```

具体实现

实现的具体方法都依赖与核心单元 `ForwardingUnit`，也就是数据转发单元。

需要进行数据转发的数据来自：`ID_EX`、`EX_MEM`、`MEM_WB` 共三个流水线寄存器；

需要接受数据转发的阶段只有：`Decode` 和 `Execution` 两个 `stage`。其中 `Decode` 需要来自 `ID_EX`、`EX_MEM`、`MEM_WB` 共三个流水线寄存器的数据，而 `Execution` 只需要 `EX_MEM`、`MEM_WB` 共两级流水线寄存器的数据。

对于上述需要转发的三个流水线寄存器，我们记录以下信息：

- 转发的数据 `forwardingData`
- 该数据本该写入目的寄存器 `id`：`regId`
- 该数据是否已经在此阶段被准备好转发：`dataReady`

其中对于 `dataReady`，比如 `lw` 指令所需要存入寄存器的数据，在 `ID_EX`、`EX_MEM` 两个流水线寄存器中是无法转备好的，所以 `dataReady` 为假，直到其需要被转发的数据从 `dataMemory` 中读出并保存到 `MEM_WB` 流水线寄存器。我们对每一个需要接受数据转发的阶段，都调用一个数据转发函数，判断是否能够成功转发（依据 `dataReady`），如果可以的话，直接更新读出的数据，这就是数据旁路转发；如果不可以的话就必须阻塞并添加气泡，直到存在冒险的数据可以从转发的数据中获取。

另一种阻塞的情况是结构冒险，在本次实验中具体体现为(`MDU` 正忙)，因为 `MUL` 和 `DIV` 等需要 `MDU` 计算的指令不包含目的寄存器，只有计算完成后，通过取指令，如 `MFHI` 和 `MFLO` 获取数据并指定保存的目的寄存器。所以该情况也需要阻塞，并且阻塞周期由计算指令所需周期决定。

处理器架构总览

```

1  TopLevel
2  |   GeneralPurposeRegister           //register file
3  |
4  |   └─InstructionFetch
5  |   |   ProgramCounter               // pc register
6  |   |   InstructionMemory            // store and fetch the hexadecimal
    |   instruction
7  |
8  |   └─InstructionDecode
9  |   |   ControlUnit                 // return control signals
10 |   |   ForwardingUnit              // Forwarding unit to eliminate Data
    |   race
11 |
12 |   └─Execution

```

```

13 |   |   ForwardingUnit           // Forwarding unit to eliminate Data
    |   |   race
14 |   |   MultiplicationDivisionUnit // Mul, div and so on
15 |   |   └─ ArithmeticLogicUnit   // add, sub, ori, xor and so on
16 |   |   |   Adder                 // carry look-ahead adder
17 |   |   |   Subber
18 |   |
19 |   └─ Memory
20 |   |   DataMemory                // Memory to store data
21 |
22 |   └─ WriteBack

```

小结

50条指令是一个不小的工程，需要处理的数据冲突情况数不胜数。为了解决这种情况，设计了 `dataReady` 信号，这也是本实验最为成功的地方，因为其可以不用枚举具体冲突情况而普适的解决所有数据冲突情况，所以极大的减少了代码量，减小了出错的可能性。其次通过处理器的设计，懂得了具体设计优劣的重要性以及掌握了 `system verilog` 语言很多细节特性，也掌握了一定 `vivado` 调试方法；最后还明白了封装的重要性，对于这么一个工程，内部信号和数据的传递都是对用户不可见的，所以数据封装很重要，我们还要利用宏，提高代码复用，减少数据修改工程量。

不足之处在于 MDU 时仍然存在结构冒险，其需要阻塞的周期较长。其中有一种情况可以优化：如果当前 MDU 正忙，但是来了新的乘除指令，那么其实可以进行覆写，减少阻塞周期。

附录

```
E:\VivadoProject\Mips\MyAns>fc 0dE.txt ../TestAns/0dE.asm.txt
正在比较文件 0dE.txt 和 ../TESTANS/0DE.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 0eC.txt ../TestAns/0eC.asm.txt
正在比较文件 0eC.txt 和 ../TESTANS/0EC.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 0hJ.txt ../TestAns/0hJ.asm.txt
正在比较文件 0hJ.txt 和 ../TESTANS/0HJ.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 0eL.txt ../TestAns/0eL.asm.txt
正在比较文件 0eL.txt 和 ../TESTANS/0EL.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 0vM.txt ../TestAns/0vM.asm.txt
正在比较文件 0vM.txt 和 ../TESTANS/0VM.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 02H.txt ../TestAns/02H.asm.txt
正在比较文件 02H.txt 和 ../TESTANS/02H.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 08H.txt ../TestAns/08H.asm.txt
正在比较文件 08H.txt 和 ../TESTANS/08H.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 22H.txt ../TestAns/22H.asm.txt
正在比较文件 22H.txt 和 ../TESTANS/22H.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 28H.txt ../TestAns/28H.asm.txt
正在比较文件 28H.txt 和 ../TESTANS/28H.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 82H.txt ../TestAns/82H.asm.txt
正在比较文件 82H.txt 和 ../TESTANS/82H.ASM.TXT
FC: 找不到差异

E:\VivadoProject\Mips\MyAns>fc 88H.txt ../TestAns/88H.asm.txt
正在比较文件 88H.txt 和 ../TESTANS/88H.ASM.TXT
FC: 找不到差异
```