# 实验八 实现I型和U型运算类指令的理想流水线设计实验

## 实验目的

1. 掌握I型和U型运算类指令的数据通路。
2. 掌握在五级流水线中添加I型和U型指令的方法。

## 实验原理与实验内容

### I型与U型运算类指令列表

本实验在实验九结构基础上继续添加I型以及U型格式的两类运算指令，构建实现R、I、U型运算类指令的理想流水线处理器。新增指令的格式如表6-1和表6-2所示。每条指令的功能说明请查阅1.1.4节。

* + - * 1. I型运算类指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 26 | 25 | 24 | 20 | 19 | 15 | 14 | 12 | 11 | 7 | 6 | 0 |  |
| **imm12** | | | | | **rs1** | | **funct3** | | **rd** | | **opcode** | | **I-Type** |
| imm[11:0] | | | | | rs1 | | 000 | | rd | | 0010011 | | addi |
| imm[11:0] | | | | | rs1 | | 010 | | rd | | 0010011 | | slti |
| imm[11:0] | | | | | rs1 | | 011 | | rd | | 0010011 | | sltiu |
| imm[11:0] | | | | | rs1 | | 100 | | rd | | 0010011 | | xori |
| imm[11:0] | | | | | rs1 | | 110 | | rd | | 0010011 | | ori |
| imm[11:0] | | | | | rs1 | | 111 | | rd | | 0010011 | | andi |
| 000000 | | shamt | | | rs1 | | 001 | | rd | | 0010011 | | slli |
| 000000 | | shamt | | | rs1 | | 101 | | rd | | 0010011 | | srli |
| 010000 | | shamt | | | rs1 | | 101 | | rd | | 0010011 | | srai |
| imm[11:0] | | | | | rs1 | | 000 | | rd | | 0011011 | | addiw |
| 0000000 | | | shamt | | rs1 | | 001 | | rd | | 0011011 | | slliw |
| 0000000 | | | shamt | | rs1 | | 101 | | rd | | 0011011 | | srliw |
| 01000000 | | | shamt | | rs1 | | 101 | | rd | | 0011011 | | sraiw |

* + - * 1. U型运算类指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 |  |  |  |  |  |  |  | 12 | 11 | 7 | 6 | 0 |  |
| **imm20** | | | | | | | | | **rd** | | **opcode** | | **U-Type** |
| imm[31:12] | | | | | | | | | rd | | 0110111 | | lui |
| imm[31:12] | | | | | | | | | rd | | 0010111 | | auipc |

### 以addi指令为例

本实验需要实现的I型和U型运算类指令与实验九中已实现的R型运算类指令的功能十分相似，都是对两个操作数进行运算，将结果写入目的寄存器中。区别在于R型运算指令的两个源操作数均来自通用寄存器堆，而本实验将要实现的指令的源操作数还可能来自指令的立即数字段或是当前指令对应的PC值。

下面以addi指令为例，介绍如何在实验九的基础上修改数据通路，从而升级处理器的功能。表6-3展示了手册对addi指令的定义。

* + - * 1. addi指令的定义

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 位数  指令 | 12位 | 5位 | 3位 | 5位 | 7位 | I 型格式的算术运算指令功能 |
| **imm12** | **rs1** | **funct3** | **rd** | **opcode** |
| addi rd,rs1,imm12 | imm[11:0] | rs1 | 000 | rd | 0010011 | 立即数加法: rs1+SEXT64(imm12)→rd |

对比手册对add和addi指令的定义可以发现，两者的区别在于addi指令将源操作数2替换为了符号拓展的立即数。因此，可以复用实验九的大部分数据通路。

前端

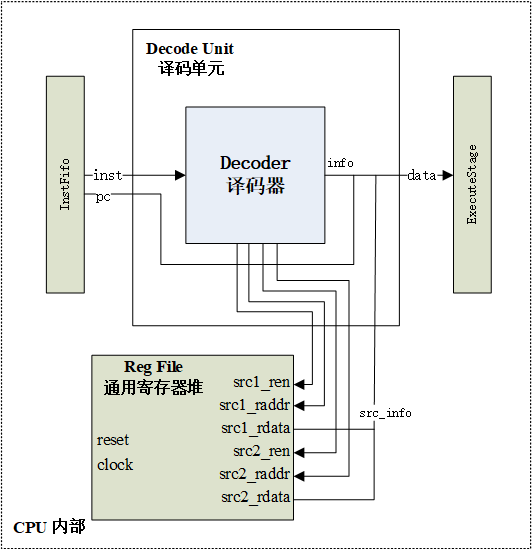
前端包括取指单元以及指令队列等部件，只负责指令的取回，因此无需进行修改。

后端

图6-3展示了目前已支持R型运算类指令的译码单元。两个源操作数被打包成src\_info数据包发往执行单元。src\_info数据包内包含src1\_data和src2\_data这两个源操作数，其被发往执行单元的ALU中进行相应的计算。ALU只会将两个源操作数根据不同的op进行不同的运算输出最终结果。因此，大部分R型计算指令的数据通路可以被复用，只需将译码单元中src2\_data的值的来源从通用寄存器堆替换为经过符号拓展的立即数的值，并且还需将译码得到的op值设置的足够精巧，这样就可以不必改动后续数据通路并实现数据通路的复用。

综上所述，只需要修改译码单元即可完成新指令的支持，需要改动的地方有三点：

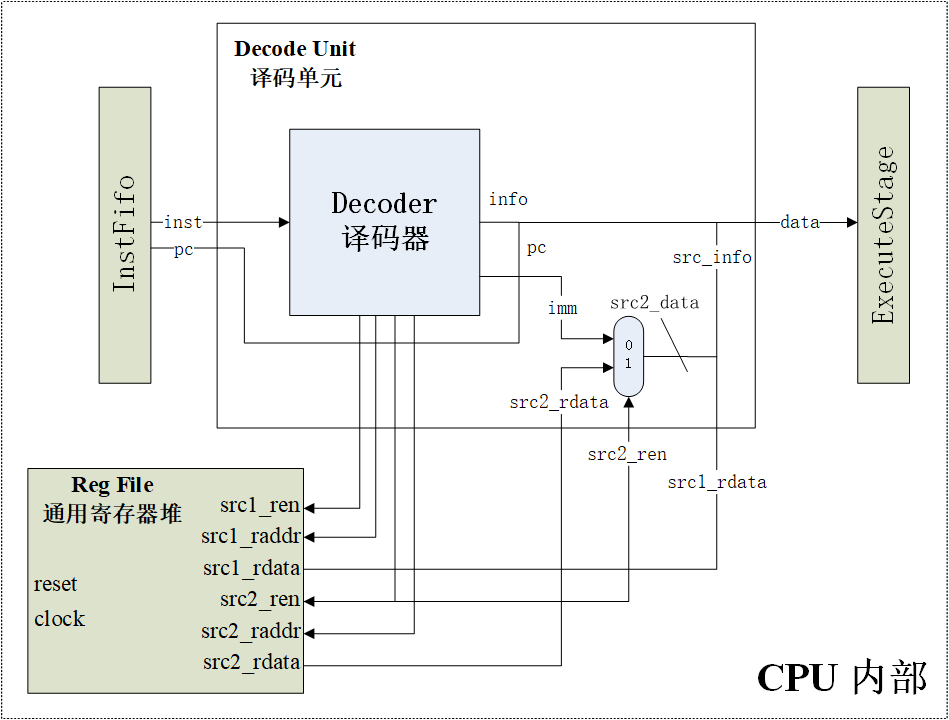
1. 需要使用指令的立即数字段说明译码器需要译码生成imm信号。
2. 源操作数来源的增加意味着需要实现一个数据选择器，使得src2\_data可以选择数据来自于通用寄存器堆还是立即数。
3. 对于op的设计，我们只需要将addi指令的op设置的与add的op一致，即可使ALU对addi指令也执行和add指令一样的操作。



* + - 1. 支持R型运算类指令的译码单元
         1. 译码信号

|  |  |
| --- | --- |
| 信号名 | 含义 |
| 已经实现的信号 | |
| src1\_ren | src1是否需要读通用寄存器堆 |
| src1\_raddr | src1的通用寄存器堆读地址 |
| src2\_ren | src2是否需要读通用寄存器堆 |
| src2\_raddr | src2的通用寄存器堆读地址 |
| op | 指令的操作类型 |
| reg\_wen | 是否需要写回通用寄存器堆 |
| reg\_waddr | 通用寄存器堆的写地址 |
| 需要增加的信号 | |
| imm | 立即数 |

已实现R型运算指令的处理器译码器产生的控制信号如表6-3所示。对于数据选择器的选择端，可以使用src2\_ren进行控制数据的选择。对于add指令而言src2\_ren为1，此时需要选择来自通用寄存器堆的src2\_rdata数据；对于addi指令而言src2\_ren为0，此时选择来自译码器的imm数据。因此只需要译码器进行修改，使得立即数指令译码得到的src2\_ren为0即可。同时译码器需要将指令的立即数字段也就是31至20位进行符号拓展作为imm信号，输出到数据选择器的选择端。



* + - 1. 经过修改支持addi指令的译码单元

修改完成的译码单元数据通路如图6-4所示。数据选择器根据src2\_ren选择数据得到src2\_data。译码单元将src2\_data和从通用寄存器读取到的src1\_rdata一起打包成src\_info数据包并与pc以及info数据包[[1]](#footnote-1)一块打包成data数据包发往下一级。

当data数据包来到执行单元后，由于addi指令的op与add指令的op一致，ALU会对addi指令的src1\_data和src2\_data同样进行add操作得到计算结果reg\_wdata。数据继续向后流动经过访存级，最后在写回级更新通用寄存器堆。

### 关于U型指令实现的建议

实验的关键在于正确的在译码单元对源操作数进行选择。需要注意的是AUIPC指令的两个操作数分别是当前指令的PC和立即数；LUI指令只有一个源操作数，但也相当于是0与立即数进行运算。

LUI指令和AUIPC指令的src1\_ren均为0，需要正确的设计数据选择器，使得LUI的src1\_data为0，而AUIPC的src1\_data为当前指令的PC值。因此src1\_data的数据选择器的数据输入端有三个分别是0、pc、src1\_rdata。可以先通过指令的opcode段判断是否为LUI指令选择0或者pc，再通过src1\_ren区分是否使用通用寄存器堆的数据。设计完成的数据通路图如图1-5所示。推荐将inst也增加到info中，这样如果在后续数据通路需要增加信号时，只需直接取用inst的内容即可不必增加更多的信号。如此处的is\_lui实际上是直接使用inst的6至0位是否等于b0110111进行赋值，写成chisel语句为：

val is\_lui = decoder.io.out.info.inst(6, 0) === "b0110111".U



* + - 1. 支持I型和U型运算类指令的译码单元

记住译码器产生的信号均在info数据包中，目前可以产生的所有信号如表6-4所示。且chisel在生成verilog时会自动删除从未使用过的信号，因此无需担心信号冗余问题。

## 实验要求

1. 在实验九的基础上继续实验，使CPU支持二、1、表6-1和二、1、表6-2中所列出的所有指令。
2. 通过本实验提供的所有仿真验证测试用例。
3. 通过本实验提供的所有板级验证测试用例。
4. 撰写实验报告：撰写报告时要求叙述以下内容，以及你对本实验的思考与探索。
   1. 选择这次实验中添加的指令中的一条，按照你自己的理解，逐步介绍其数据通路设计的思路以及实现过程。
   2. 更新上一实验中绘制完成的数据通路图。
   3. 谈谈你对数据通路复用的理解。

## 实验步骤

1. 实验步骤参见实验九的实验步骤部分
2. 提醒与建议
   1. 在chisel中有多种数据选择器类型，如Mux、MuxCase、MuxLookup、Mux1H等，请参考chisel教程章节选择适合的数据选择器。
   2. 当reg\_ren为0时，建议将reg\_raddr置为0，防止CPU出现意想不到的状况。

## 思考与探索

1. 为什么在RISC-V指令中没有SUBI指令？
2. 观察二、1、表6-2中指令的imm字段，为什么imm字段的长度被设计为20位？请问这样设计可以和哪些指令搭配使用并发挥什么样的效果？
3. 谈谈你在实验中碰到了哪些问题？又是如何解决的？

1. info数据包内包含op、reg\_wen和reg\_waddr [↑](#footnote-ref-1)