## 数据通路

控制器在一个时钟周期内只能发出一组控制信号

一个部件在一个时钟周期内只能完成一个操作，所以单周期CPU数据通路必须采用部件冗余计数来构建

已知一个指令周期分为多个阶段，在不同的阶段可能会多次使用同一个或同一类部件，比如，取指阶段、数据读取阶段，存放结果阶段都可能访问存储器；

使用部件冗余技术和部件共享技术，两种方式处理这种情况

部件冗余技术：由于单周期CPU的指令周期等于时钟周期，那么在一个时钟周期内，就要完成取指、译码、执行指令等各个阶段的工作，在这些阶段中，可能会使用到同一个或同一类的部件，但另一方面，各个阶段又处于同一时钟周期，再同一时钟周期内，很多部件只能访问一次，所以只能设置多个同种类的部件分别支持各个阶段的操作。

以执行两条指令I1和I2为例，I1在时钟周期T1执行，I2在时钟周期T2执行，

指令I2在IM中的地址是在指令I1取出(顺序执行) 或执行(跳跃执行)之后 形成的

对于单周期CPI来说，将I2的地址打入PC是在时钟周期T2的上升沿完成的，并且，若指令I1的执行结果要存入RF的话，也是在时钟周期T2的上升沿完成的

改进：

某些功能单元的数据来自两个不同的源头，例如，写入PC的值可能来自两个加法器中的任一个，写入RF的数据可能来源于ALU或者DM，但这些不同来源的数据线不可能简单地拼接在一起，必须增加多路选择器MUX用来从多个数据源中选择其中一个传输给目的单元

I型指令的立即数是16位，而ALU是32位运算单元，所以，来自指令的16位立即数需要经过符号扩展后输入到ALU的一端。因此需要增加符号扩展单元（SigExt16/32）

J型指令形成转移地址时需要进进行左移2位的操作，所以需要增加移位器SHL2

改进：

还缺少一些MUX和其他功能部件：如R型指令中第15~11位是目的寄存器号Rd，而取数指令中第20~16位是源寄存器号Rt，这样，在IM和RF间就需要设置一个二路选择器，在指令的第15~11位和第20~16位之间选择一路作为写入寄存器编号

PC值有三种修正方法，即下一条指令地址的计算方法：

对于非转移指令，在取值阶段执行PC+4

对于分支指令，条件满足时，将PC+4的高4位凭借后形成32位转移地址

对于转移指令J，将Addr26左移两位与PC+4的高4位拼接后形成32位转移地址

对着三路PC修正值的选择可以用两个二路选择器或一个四`路选择器实现

对于ALU的功能选择控制可以从主控制单元中分离出来，采用分散式控制器设计方案，即将控制单元分为两个部分，主控制单元MCU和ALU控制单元ALUCU，

MCU的输入包括指令操作码OP\_Code和主时钟信号CLK，输出包括输入到ALUCU的ALU操作命令ALUOp，ALUCU再根据该操作命令和R型指令中的Func字段产生ALU功能选择控制信号

分支指令BEQ分两种情况进行，条件成立和条件不成立，差异是写入不同的PC值

所以执行BEQ指令是，主控制单元首先产生一个专门的控制信号Branch，

是否实现转移取决于ALU的Z输出状态，Z=1表示条件成立，将转移地址写入PC，Z=0表示条件不成立，将PC+4写入PC

Inst[]代表一个指令字，如Inst[25-0]表示指令字中的第25位到0位

## 信号传输

|  |  |
| --- | --- |
| 传输方向 | 传输内容 |
| PC→IM(Addr) |  |
| IM(Inst)→RF(R\_Reg1) | Inst[25-21] |
| IM(Inst)→RF(R\_Reg2) | Inst[20-16] |
| IM(Inst)→RF(R\_Reg) | Inst[20-16]或Inst[15-11] |
| IM(Inst)→ALUCU | Inst[5-0] |
| IM(Inst)→Add2 | Inst[15-0] |
| RF(R\_data1)→ALU(A) |  |
| RF(R\_data2)→MUX2 |  |
| IM(Inst)→MUX2 | Inst[15-0] |
| MCU→MUX2 | ALUSrc |
| MUX2→ALU(B) |  |
| (PC)→Add1\_B |  |
| (PC)+4→(PC) |  |
| Inst[25-21]→R\_Reg1(RF)，(R\_data1)→ALU\_A  Inst[20-16]→R\_Reg2(RF)，(R\_data2)→ALU\_B  Inst[15-11]→R\_Reg(RF)， |  |
|  |  |
|  |  |
| (R\_data1)→ALU\_A |  |
|  |  |

## 指令周期

|  |  |
| --- | --- |
| 取指令 | (PC)→IM(Addr)  Read(IM)  (PC)→Add1\_B，(PC)+4→(PC) |
|  |  |
| 指令译码 |  |
| 读取数据 |  |
| 执行操作 |  |
| 存放结果 |  |

## CPU性能

|  |  |  |
| --- | --- | --- |
| 操作延迟时间 | 寄存器数据建立时间 | t\_setup |
| 寄存器数据读出时间 | t\_cpq |
| 通用寄存器组数据建立时间 | t\_RFsetup |
| 通用寄存器组数据读出时间 | t\_RFread |
| 移位器延迟时间 | t\_left |
| 存储器读写时间 | t\_mem |
| ALU运算延迟时间 | t\_ALU |
| 符号扩展延迟时间 | t\_sex |
| 多路开关延迟时间 | t\_MUX |

各指令执行时间

|  |  |
| --- | --- |
| ALU | T=t\_cpq-t\_pc+t\_mem-t\_IM+t\_RFread+t\_ALU+t\_MUX+t\_RFsetup |
| LW | T=t\_cpq-t\_pc+t\_mem-t\_IM+t\_RFread+t\_ALU+t\_mem-t\_DM+t\_MUX+t\_RFsetup |
| SW | T=t\_cpq-t\_pc+t\_mem-t\_IM+t\_RFsetup+t\_ALU+t\_mem-t\_DM |
| BEQ | T=t\_cpq-t\_pc+t\_mem-t\_IM+t\_RFsetup+t\_MUX+t\_ALU+t\_MUX+t\_MUX+t\_setup-t\_pc |
| J | T=t\_cpq-t\_pc+t\_mem-t\_IM+t\_left+t\_MUX+t\_setup-t\_pc |

时钟周期

定长指令周期

时钟周期=指令周期=执行时间最长的指令所需时间

T=t\_cpq-t\_pc+2t\_mem+t\_RFread+t\_ALU+t\_MUX+t\_RFsetup