LC-3流水线处理器设计

郭家华 PB10000614

李博杰 PB10000613

2012年06月09日

Contents

1 总体设计

1.1 时序

处理器时钟来自于信号线clk，clk为方波。记clk上升沿为clk.p，下降边沿为clk.n。以clk.p为界，划分的时间片称为周期。ckl.n将一个周期分为上半周期和下班周期。

本处理器使用三级流水线，分别为取值+译码（S0）、运算（S1）、访存（S2），每一级流水线占一个周期。 每个周期分为上半周期（a）、下半周期两个部分（b），因此一共有六个半周期（子周期）：S0.a S0.b S1.a S1.b S2.a S2.b。 取指的访存操作发生在S0.b，数据内存的读写发生在S2.b。

有时处理器需要清空流水线，即阻止、还原当前流水线中所有指令对寄存器和数据内存的写入。 是否清空流水线由处理器的内部信号线flu决定，flu信号在S2.a产生。

1.2 指令分类

指令分为R-type、I-type、J-type三类，I-type又分为load系列和store系列。 I-type指令中的内存间接寻址指令（LDI、STI）暂不考虑。

类型	写寄存器	读内存	写内存	改变PC
R-type	yes	no	no	no
load	yes	yes	no	no
store	no	no	yes	no
J-type	no	no	no	yes

1.3 流水段寄存器

指令，以及由指令解码出的一些值，被放入流水段寄存器中。各个阶段的具体行为受流水段寄存器的控制。

指令内存输出的指令经decoder解码产生流水段s0（纯组合逻辑）。 s0在下一个clk.p时流入S1阶段的s1，s1在下一个clk.p时流入S2阶段的s2， s2在下一个阶段流出流水线。

大意如下：

```
always@(posedge clk)
begin
    s1 <= s0;
```

```
    s2 <= s1;
end
```

如果考虑flu，则为：

```
always@(posedge clk)
if(flu) begin
    s1 <= noop_state;
    s2 <= noop_state;
end else begin
    s1 <= s0;
    s2 <= s1;
end
```

其中，noop_state对应的操作不写寄存器、不写内存、不修改PC。

流水段寄存器包含以下内容：

名称	含义
pc	下一条指令的地址
op	指令高4位
dra	目标寄存器地址
sr1a	源寄存器1地址
sr2a	源寄存器2地址
iswb_alu	指令在S1阶段写寄存器
iswb_mem	指令在S2阶段写寄存器
imm	指令包含的立即数
alu_ops	alu进行的运算类型（add、and、not）
c_sr1	alu的输入1是sr1还是pc
c_sr2	alu的输入2是sr2还是imm（立即数）
sjnzp	{ 跳转时是否将pc保存至r7；是否无条件跳转；n、z、p=1时是否跳转 }
memw	是否写内存
apsr_is_alu	指令在S1阶段更新psr
apsr_is_mem	指令在S2阶段更新psr
load_mem	指令是否读内存
indirect_mem	是否有内存间接寻址操作（LDI，STI）
trap	trap指令

1.4 寄存器设计

寄存器堆（greg）由八个单寄存器（sreg）加上相应的地址译码电路构成。

sreg中有两个16位寄存器d1和d2。 在时钟边沿，如果由运算阶段流水至访存阶段的指令为R-type指令， 则alu的输出被写入d1，否则d1值不变。 如果从访存阶段流出的指令为load系列指令，则数据内存读出的数被送入d2中， 否则d2的值被置为d1的值。即：

```
always@(posedge clk)
if(we1)
    if(we2)
    begin
        d1 <= alu_out;
        d2 <= mem_out;
    end else begin
        d1 <= alu_out;
        d2 <= alu_out;
    end
else if(we2)
    begin
        d1 <= mem_out;
        d2 <= mem_out;
    end else begin
        d2 <= d1;
    end
```

或者写成：

```
always@(posedge clk)
begin
    d1 <= we1 ? alu_out : we2 ? mem_out : d1;
    d2 <= we2 ? mem_out : d1;
end
```

其中，对与R-type指令，we1为1，否则为0。对于load类指令，we2为1，否则为0。

如果考虑flu信号，则：

```
always@(posedge clk)
if(flu)
    d1 <= d2;
else begin
    d1 <= we1 ? alu_out : we2 ? mem_out : d1;
    d2 <= we2 ? mem_out : d1;
end
```

1.5 跳转与清空流水线

一条指令执行的结果可能是写寄存器、写内存或改变PC的值。 如果一条指令最终改变了寄存器或内存的内容（写入新值），我们称这条指令交付了。

当发生跳转或异常或外部中断时，指令将不再顺序执行。我们将这个改变执行流的指令称为J指令。J指令自生不交付。

当J指令流出流水线时，处理器才实际进行跳转，而此时流水线中已有一些顺序跟在J指令之后的、 事实上不应该执行的残余指令，我们称这些指令为尾指令。尾指令最终不会交付，因此尾指令最终不能修改内存、寄存器。

实现：

数据内存

对于S0、S1中的指令：指令尚未进入访存阶段(S2)，因此将S0、S1中的指令替换成空指令即可。

对于S2中的指令：flu信号产生于S2.a，因此，在S2.b中写内存时考虑flu信号即可屏蔽S2中指令的写内存操作。

寄存器堆

对于S0中的指令：指令尚未进入运算阶段(S1)和访存阶段(S2)，因此将S0中的指令替换成空指令即可。

对于S1中的指令：运算结果尚未提交至d1，在clk.p时阻止寄存器写即可。详见1.4。

对于S2中的指令：该指令在S1、S2阶段都有可能写寄存器。若打算在S2阶段写寄存器，在clk.p时阻止寄存器写即可；若已在S1阶段写寄存器，在clk.p时将d1还原成d2即可。详见1.4。

1.6 数据相关

说明：

1. 这里的数据相关是指围绕寄存器的相关，当前设计不存在关于数据内存的相关
2. 寄存器取数均发生在S1阶段
3. 若上一周期S1阶段写所写寄存器为本周期所读寄存器， 则本次取出的数是上一周期S1阶段刚写入的数

1.6.1 相关检测

我们简单将指令分为四类（暂不考虑内存间接寻址）：

类型	S1.read	S1.write	S2.write
1	yes	yes	no
2	yes	no	yes
3	yes	no	no
4	no	no	no

其中Si.read/write表示指令在Si阶段要读/写寄存器。

R-type（add、and、not）为类型1，load系列指令为类型2，store系列指令为类型3。 无条件跳转指令为类型3或4，条件跳转指令为类型4。

现在考虑相邻的两条指令：

前一条指令类型	后一条指令类型	是否相关
1	any	no
2	1 or 2 or 3	maybe
3 or 4	any	no

以上结论显然。

由此可知，前一条指令为load类时，才有可能相关，相关条件是load写的寄存器为后一条指令读的寄存器，即：

```
( S1.sra1 == S2.dra || S1.sra2 == S2.dra )
```

1.6.2 相关处理（插气泡）

处理方式即暂停流水线，插入气泡。

将1.3中的流水代码修改成这样：

```
always@(posedge clk)
if(flu) begin
    s1 <= noop_state;
    s2 <= noop_state;
end
else if(bubble)
    s2 <= noop_state;
else begin
    s1 <= s0;
    s2 <= s1;
end
```

其中bubble信号线指示处理器是否需要暂停流水线，信号在clk.p时产生。

另一方面，当bubble=1时阻止通用寄存器、psr寄存器的写入。

这样一来，S1中的指令执行了两次，第一次不写寄存器。第二次执行时， load指令已将数据写入寄存器堆了。

1.7 复杂指令实现

复杂指令是指无法通过顺序流水实现其功能的指令，包括LDI、STI等间接寻址指令和中断隐指令等。 其中，LDI、STI执行过程需要两次访存。

复杂指令实现的基本思路是：在逻辑上将单个复杂指令拆分成若干个简单指令， 并阻塞流水线、占用多个周期处理单条复杂指令。 如，LDI可拆分成LDR+LDR，STI可拆分成LDR+STR。

现针对LDI、STI这两条特殊指令设计解决方案。

1.7.1 间接寻址指令实现

这两条指令较之与LDR、STR，多了一次访存操作。因此，暂停S1及之前的流水线一个周期、让该指令在S2阶段多盘旋一个周期，即可。

指令在S2的第一个周期时，从内存中读出有效地址存于一个临时寄存器中；在S2的第二个周期时，以临时寄存器中的值为地址送入数据内存，执行实际的读写操作。

1.7.2 LDI的相关处理

在1.6.1中已说明，load类指令后接其他指令可能会有数据相关， 需要插气泡、暂停流水线。 所以LDI指令后接其他指令，也有可能产生数据相关。

因为LDI实际写寄存器发生在第二阶段，所以如果有相关， 在LDI自生暂停流水线的基础上还要再多暂停一周期，插入bubble。

1.8 暂停流水线

流水线在两种情况下会暂停：

1. 数据相关??
2. 执行复杂指令??

详情请参考相关内容。