# 基于 MIPS 指令集的流水处理器设计

——计算机组成原理课程设计

08 级计算机科学与技术 B 班
作者：张华志　08380131
指导老师：李国桢
2010 年 1 月 6 日

# 基于 MIPS 指令集的流水处理器设计

08 级计算机科学与技术 B 班　　08380131　　张华志

## 1. 选题背景

在计算机组成原理课程中，我学到了很多关于计算机的五个基本部件的知识。但我还希望能够更多地了解这些部件是怎样合作，共同工作的，处理器是怎样执行指令的，控制信号又是怎样起作用的。通过对 David A. Patterson 和 John L. Hennessy 所著的《计算机组成与设计》的学习，我了解了从单周期到多周期，再到流水线 CPU 的数据通路和控制信号的设计方法。我惊奇地发现，处理器的构造竟然如此复杂和精密。在了解了这些知识后，我认为自己有能力，有兴趣去自己设计一个 CPU。

我的 CPU 的两大特点是使用 MIPS 指令集和流水线技术，这是因为我希望能够设计一个结构简化但性能优良的 CPU。另外一个原因是，尽管这两种技术对于提高计算机性能十分重要，但它们在白中英的教材中却很少提及，所以我认为自己应该设计一个基于 MIPS 指令集的流水 CPU 来增强对这两种概念和技术的理解。

流水线不是通过减少某条指令的执行时间来改善性能的，而是通过增加同时运行的程序的数量来增加数据吞吐量。 在设计流水处理器时，我们必须要考虑应该选择怎样的指令集，因为一个不合适的指令集会让设计变得更加困难。比如，某些指令集中的指令在执行时会需要一系列的内部互锁（interlocks），而这是提升性能的主要障碍，因为在执行这些指令的过程中要利用到计算机的多个模块，占用大量的时间，并因此限制时钟频率。而 MIPS 指令集的主要特点就是把所有的子过程，包括 cache 的存取，集中在一个时钟周期内。这样就避免了内部互锁，让指令能在单周期内执行完成，尽可能减少流水线运行中的各种相关问题（hazards）。

## 2. 技术路线

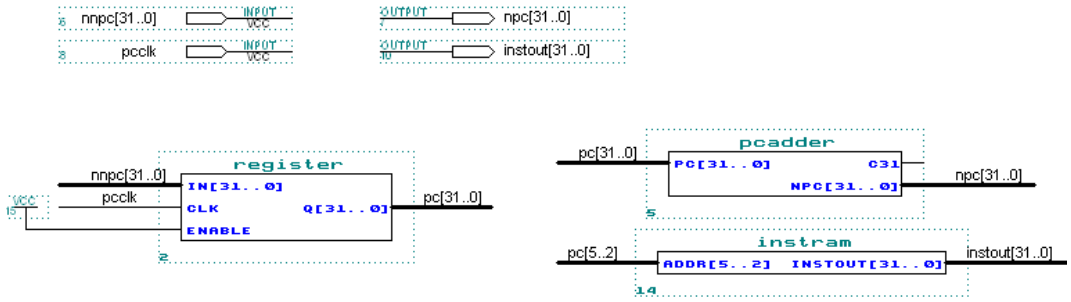
基于 MIPS 指令集的流水线是一个五级流水线，指令在流水线中依次前进。因为直接设计流水处理器十分困难而且容易出错，所以我决定先设计一个单周期的 CPU，再通过添加流水线寄存器（pipeline registers），定向传送路径（forwarding path）， 数据相关检测（data hazard detection）and 和控制相关的控制信号（flushing instructions on branch hazards）来建立最终的流水处理器。.

我选择 MAX+PLUS II 作为设计，编译和时序仿真的软件。在器件的设计上，我使用了 VHDL 设计和原理图连线设计两种方法。我用 VHDL 设计了控制单元（control unit）和定向传送控制单元（forward logic），用原理图连线法设计了其他部件。我的设计大部分是根据 David A. Patterson 和 John L. Hennessy 所著的《计算机组成与设计》来完成的。另外，朱子玉和李亚民所著的《CPU 芯片逻辑设计》和李玮超同学的论文《基于 MIPS 指令集的 32 位 RISC 处理器逻辑设计》也给了我一定的帮助。

## 3. 设计过程

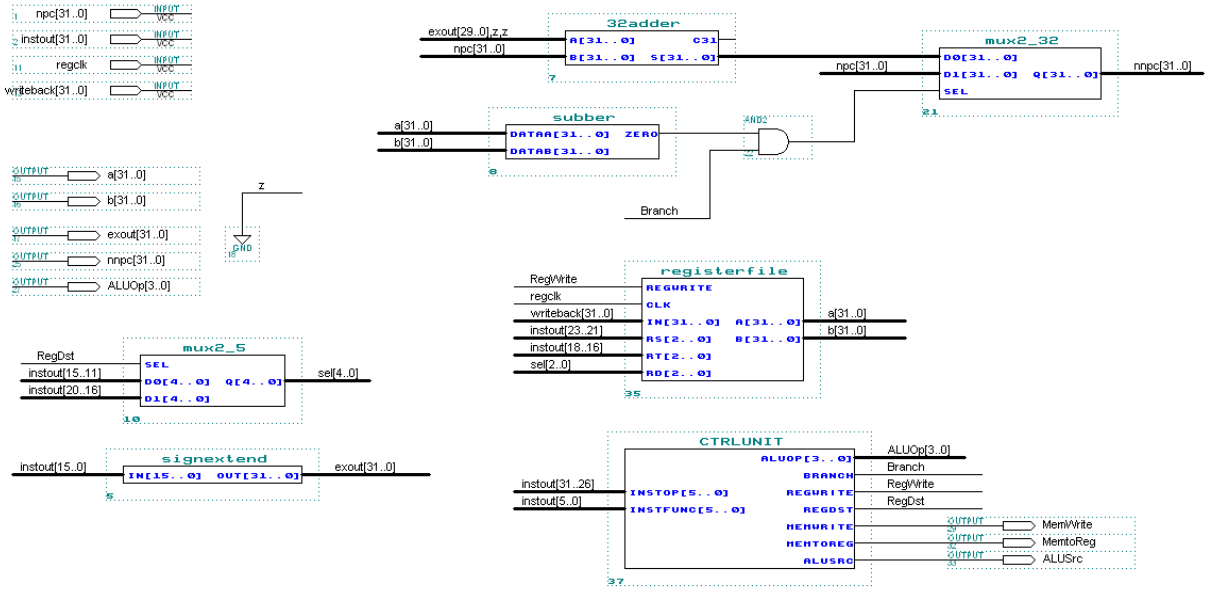
**第一步**: 构建单周期 CPU 的数据通路和控制单元。我把单周期的数据通路分成五个阶段设计，这样可以更方便地转化成流水 CPU。

(1) IF(Instruction Fetch) 阶段




PC (program counter) 是一个 32 位的在每个脉冲后被写入的寄存器。PCADDER 是一个 32 位的加法器，用来计算 PC+4 的值。INSRAM 是一个只读的指令存储器，不需要时钟控制和使能信号。我使用了 LPM 器件库中的 RAM 来实现存储器的功能。

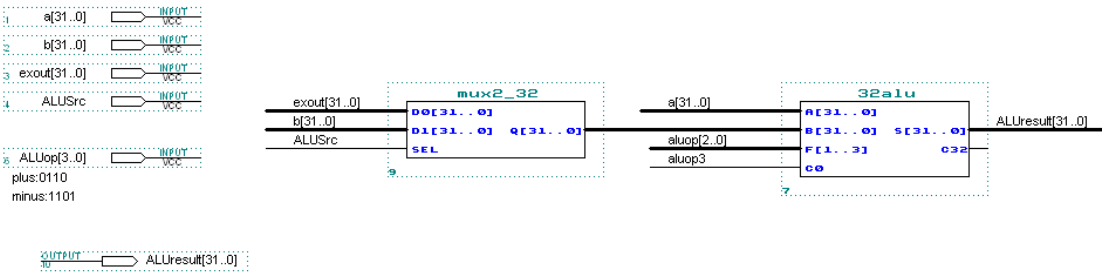
(2) ID (Instruction Decode) 阶段




REGISTERFILE 本应该是 32 个寄存器组成的寄存器堆，但如果这样设计会使编译非常困难，所以在实际设计中，我把寄存器的数目减少到 8 个，同时用原来的寄存器的 5 位选择信号的低三位来选择寄存器。为了处理立即数，还需要一个符号扩展单元，这个扩展单元的输出有两个去向，一是作为 SW 和 LW 指令中的相对地址。二是在左移两位后作为转移时相对 PC 的偏移地址。右上方的减法器是用来判断寄存器中读出的两个数是否相等的，判断的结果和控制器发出的 branch 信号一起决定是否要执行转移指令。

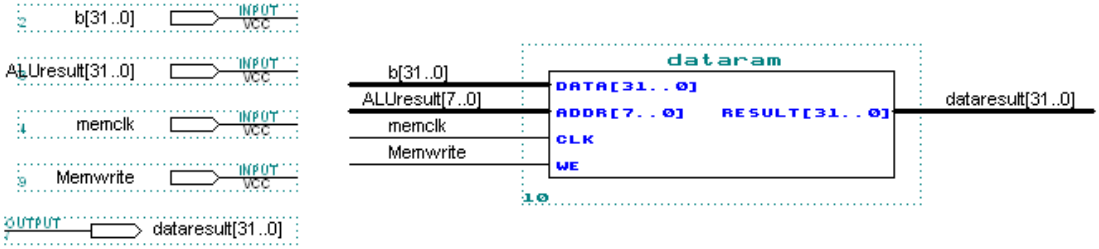
控制单元（control unit）很难用连线图的方式实现，所以我就使用 VHDL 去设计。因为我的设计思路更倾向于系统的结构而不是功能的数量，所以我的控制单元只处理 5 种指令，ADD, SUB, LW, SW, 和 BEQ。具体的 VHDL 代码请见附录 A 。

(3) EX (Execute) 阶段



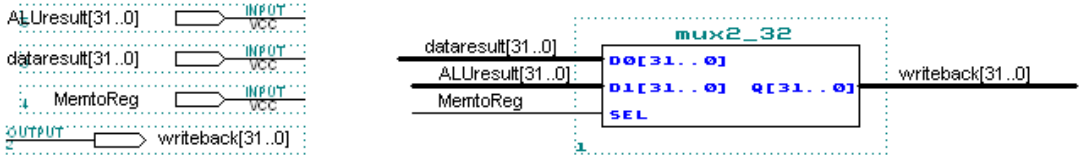

ALU 是 32 位的，ALU 的功能是受 4 位 ALU 操作信号的控制的。B 输入端的值是经过一个 32 位多路器选择的，ALUSRC 控制信号决定了这个多路器是选择符号扩展后的立即数还是 RT 寄存器的值。

(4) MEM (Memory Access) 阶段



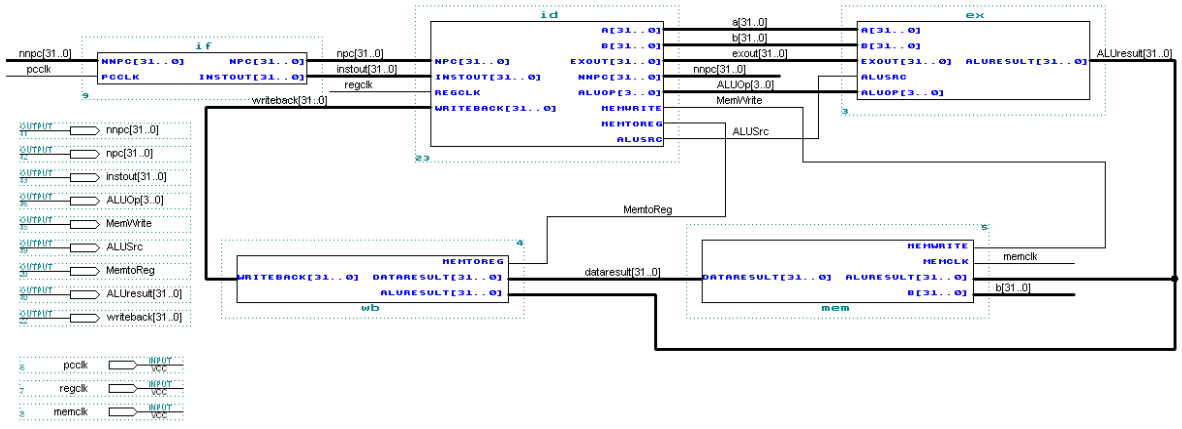

数据存储器是一个时序逻辑单元，有着地址端和写入数据端，还有一个读数据输出端。写控制信号和时钟信号对于写数据是必要的。

(5) WB (Write Back) 阶段




MUX 2_32 决定了是 ALU 的运算结果还是数据存储器的读出值作为写回的数据。MEMTOREG 信号是这个多路器的选择信号。.

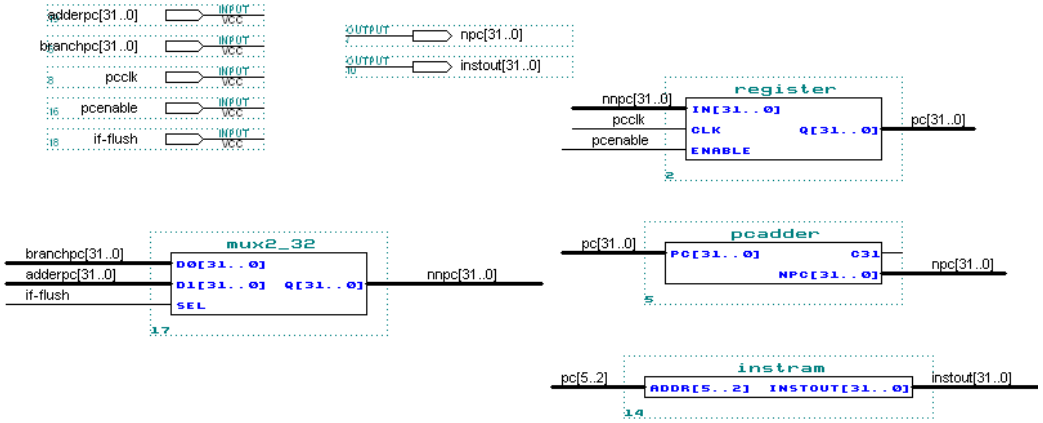
(6) 完整的单周期 CPU 连线图



**第二步：**把单周期 CPU 转变成流水线 CPU

我仍然会分五个阶段来描述构造的过程，但每个阶段只是描述比之前的单周期设计多出的部分。我会侧重描述各种相关问题的解决方法以及流水线的特点。

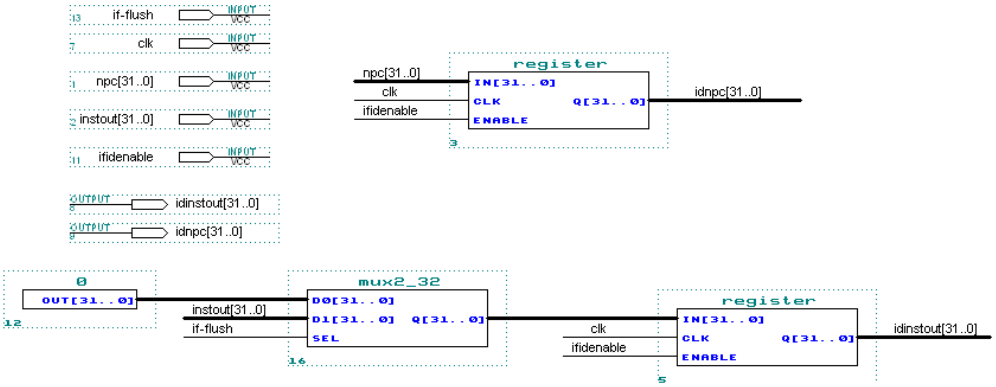
(1) IF(Instruction Fetch) 阶段




增加了一个 MUX 2_32 用来选择下一个 PC 的值是条件转移后的地址还是下一条指令的地址。这个多路器受 IF-FLUSH 信号的控制。尽管这是一个很小的改变，但却是解决控制相关（branch hazards）问题所必须的。

PCENABLE 信号能够阻止 PC 的值改变，从而帮助通过延迟来解决数据相关问题（resolve data hazards by stalling）。

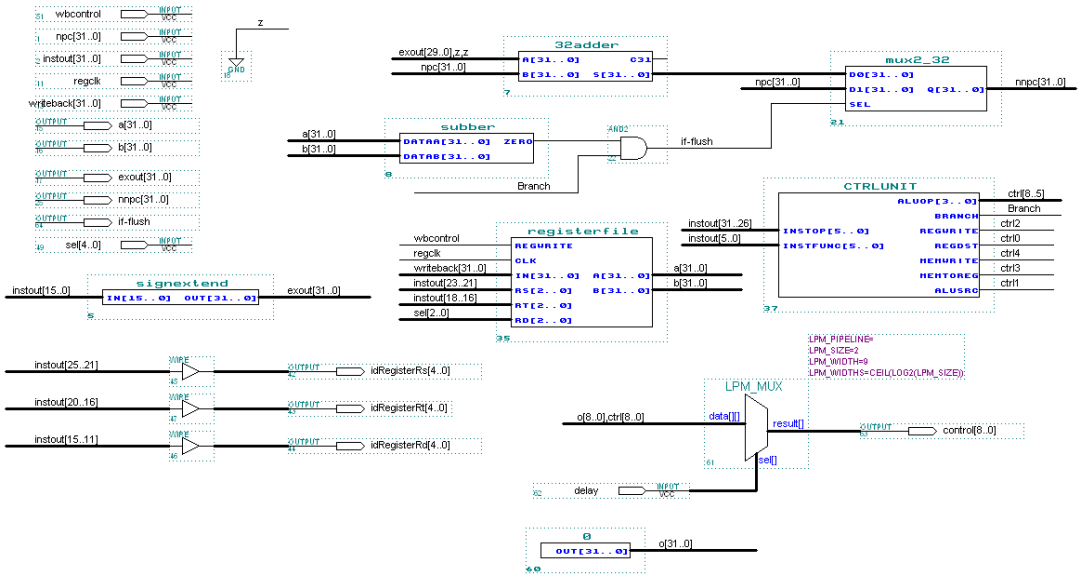
(2) IF/ID 流水线寄存器




流水线寄存器（pipeline register）是用来暂时存放一条指令的数据和控制信号的。整个流水数据通路需要 4 个这样的寄存器：IF/ID, ID/EX, EX/MEM 和 MEM/WB。流水线寄存器把各个流水阶段分隔开，除了 IF/ID 之外，其他三个流水线寄存器都由只由寄存器构成。所以我就不展示这三个寄存器的原理图了。IF/ID 寄存器之所以跟其他三个不同是因为它是通过延迟解决数据相关问题的重要环节。当需要延迟的时候 IFIDENABLE 信号就会被置 0，从而阻止 IF /ID 中寄存器的值改变。

IF/ID 寄存器不只能处理数据相关问题，还能在解决控制相关问题时发挥作用。我使用了延迟转移法（Assume Branch Not Taken scheme）来解决控制相关问题。IF-FLUSH 信号能将读取到得指令清零，从而在条件转移发生时制造一个“气泡”。

(3) ID (Instruction Decode) 阶段

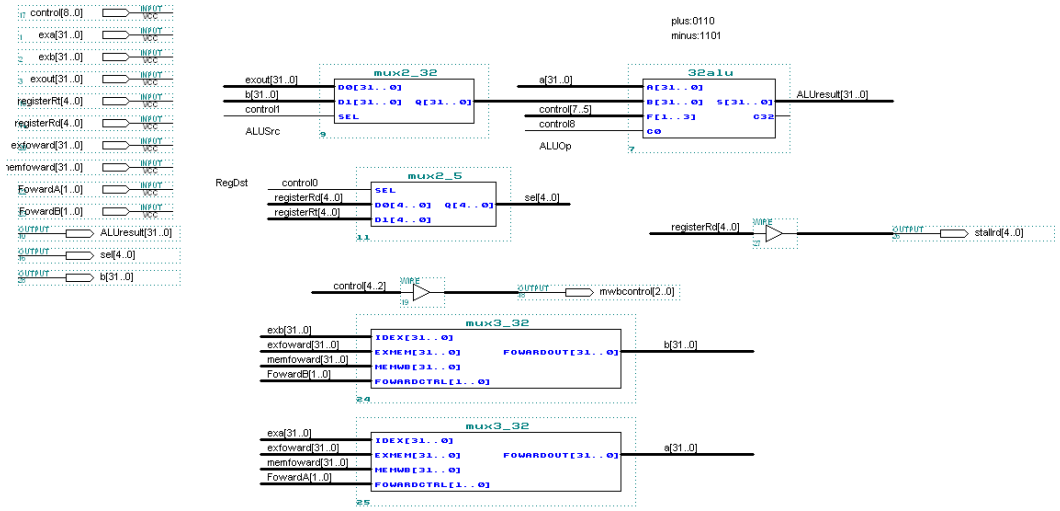



在这一阶段改动较大。

首先，RS, RT 和 RD 值和控制单元产生的控制信号应当被传递到下一个阶段，因为这些数据或信号会被定向传送单元（Forward Logic）使用，而且会在写回阶段被用到。从这一点上，我们就可以看出单周期和流水线 CPU 的一个主要差别，那就是我们必须在每个阶段把后面的阶段需要的数据和控制信号传递下去。因为不同于单周期 CPU 的是，在流水 CPU 中，我们不能在一个时钟周期后在原来的地方取到原来的数据，原来的数据会被下一条指令的数据替代。

第二，我在右下角添加了一个 MUX 2_9。这对于通过延迟解决数据相关问题是必须的。当需要延迟时，延迟信号会被置 1，所有的 9 个控制信号全部变为 0。通过我前面的描述可以看出，三个操作（把 PC 的使能信号置零，把.IF/ID 的使能信号置零，把所有的控制信号变为 0）可以使数据通路中产生一个气泡，从而通过延迟解决数据相关问题。

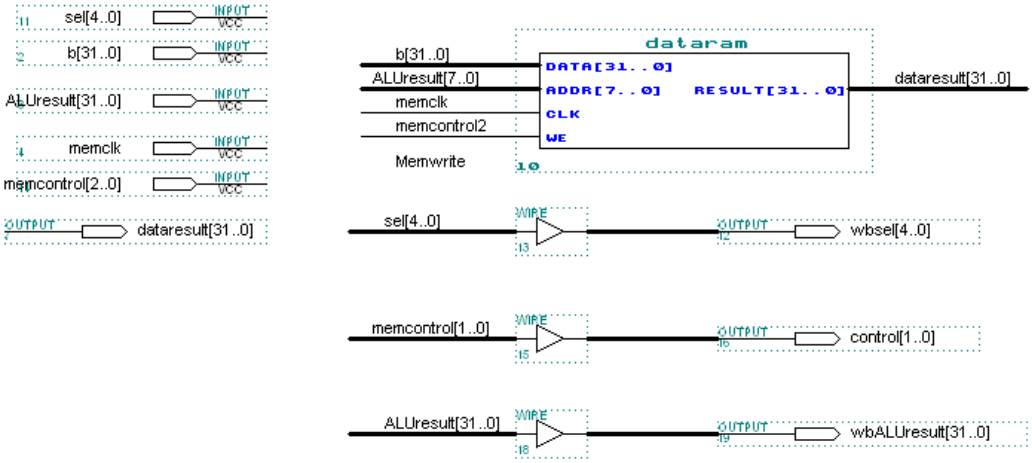
(4) EX (Execute)阶段




EX 阶段是数据相关问题频繁发生的地方。我们可以通过定向转移（forwarding）来解决这个问题。.定向转移控制单元的设计会在后面的部分单独展示。要用两个 MUX 2_32 来选择 ALU 两个端口的输入值是寄存器中读出的数据，还是两个传送数据中的

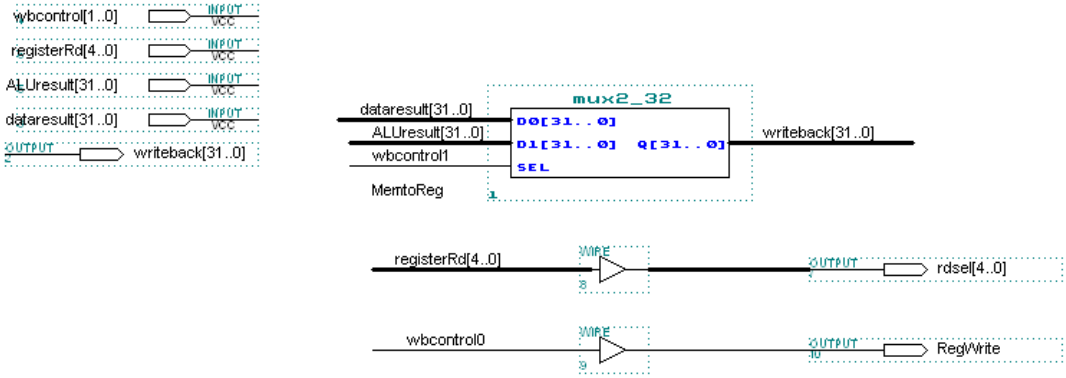
一个。就像我在 ID 阶段中做的一样，我把后面要用到的控制信号和数据传送到下一个阶段

(5) MEM (Memory Access)阶段



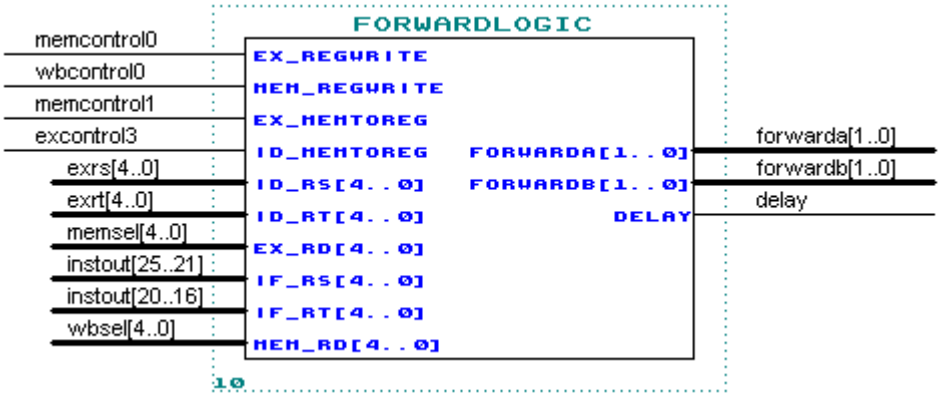

这一阶段的变化很少，只是增加了把必要的控制信号和数据传送到 WB 阶段的部分。

(6) WB (Write Back) 阶段




这一阶段同单周期的相同阶段基本相同。

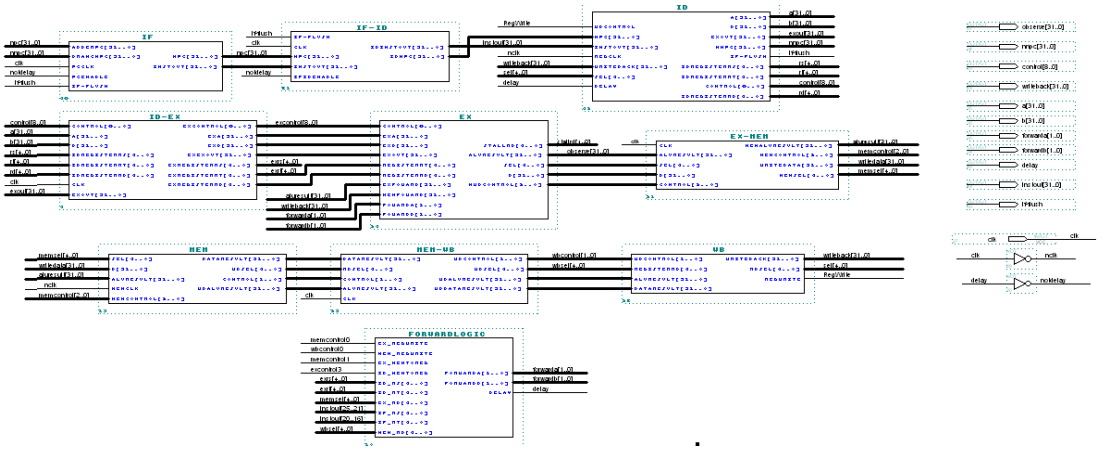
(7) 定向传送控制单元（Forward Logic）的设计




定向传送控制单元有两个功能，第一个是产生定向传送信号来通过定向传送解决数据相关问题。另一个是产生延迟信号，来通过延迟解决数据相关问题。它需要很多输入端，包括特定阶段的特定控制信号和特定寄存器的值。设计这一单元的 VHDL 代码请

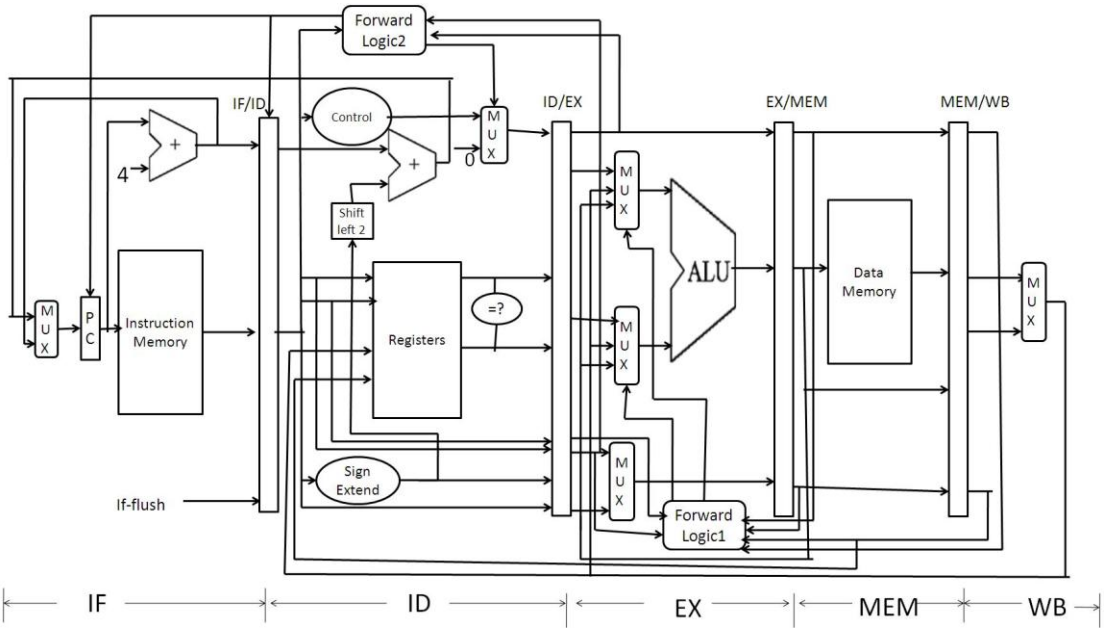
见附录 B。

(8) 流水 CPU 完整的数据通路和控制信号



因为空间的限制，我不能清楚的展示出来，这个原理图的源文件是“pipeline.gdf”.

整个流水处理器的结构如下图所以（省略了部分控制信号）




**第三步：**模拟仿真

(1) 指令序列

我的指令序列包括 5 种指令：LW, SW, ADD, SUB 和 BEQ。我精心地设计了他们的顺序使得各种数据相关问题和控制相关问题都能在仿真时被检测到。

| MIPS 指令序列 | 寄存器 | | | | |
|---|---|---|---|---|---|
| | $1 | $2 | $3 | $4 | $5 |
| 00 Lw $1, 20($0) 35 16 17 20 | 6 | 0 | 0 | 0 | 0 |
| 04 Lw $2, 21($0) 35 16 18 21 | 6 | 5 | 0 | 0 | 0 |
| 08 Lw $3, 22($0) 35 16 19 22 | 6 | 5 | 6 | 0 | 0 |
| 0C Lw $4, 23($0) 35 16 20 23 | 6 | 5 | 6 | 2 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 Lw $5, 24($0) 35 16 21 24 | 6 | 5 | 6 | 2 | 1 |
| 14 Sub $2, $1, $4 0 17 20 18 0 34 | 6 | 4 | 6 | 2 | 1 |
| 18 Add $4, $2, $5 0 18 21 20 0 32 | 6 | 4 | 6 | 5 | 1 |
| 1C Lw $2, 20($1) 35 17 18 20 | 6 | 2 | 6 | 5 | 1 |
| 20 Add $4, $2, $5 0 18 21 20 0 32 | 6 | 2 | 6 | 3 | 1 |
| 24 Beq $1, $3, 12 4 17 19 3 | 6 | 2 | 6 | 3 | 1 |
| 28 Add $0,$0,$0 0 0 0 0 0 32 | 6 | 2 | 6 | 3 | 1 |
| 2C Add $0,$0,$0 0 0 0 0 0 32 | 6 | 2 | 6 | 3 | 1 |
| 30 Add $0,$0,$0 0 0 0 0 0 32 | 6 | 2 | 6 | 3 | 1 |
| 34 Sw $4,50($3) 43 19 20 50 | 6 | 2 | 6 | 3 | 1 |
| 38 lw $1,50($3) 35 19 17 50 | 3 | 2 | 6 | 3 | 1 |
| 3C Add $0,$0,$0 0 0 0 0 0 32 | 3 | 2 | 6 | 3 | 1 |

(2) 生成 “MIF”文件来初始化内存

因为在数据通路中有一个数据存储器和一个指令存储器，所以需要两个 MIF 文件。我分别命名为 “dataram.mif”和“instram.mif”。

Dataram.mif:

```
WIDTH = 32;
DEPTH = 256;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
     0    :    00000000;

    ......

--   :    --------;
14   :    00000006;
15   :    00000005;
16   :    00000006;
17   :    00000002;
18   :    00000001;
19   :    00000000;
1a   :    00000002;
1b   :    00000000;

    ......
```

Instram.mif:

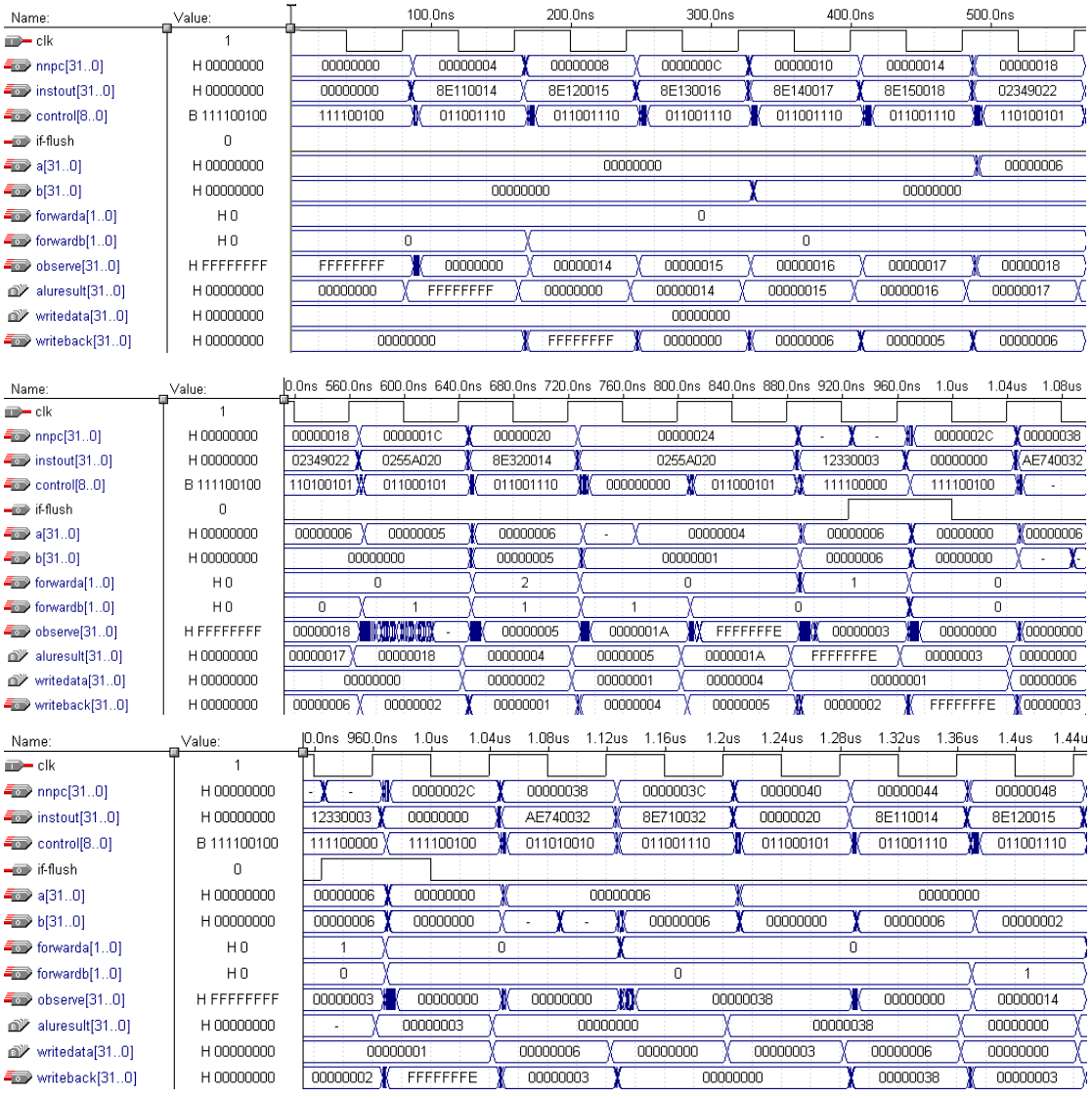
```
WIDTH = 32;
DEPTH = 16;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
        0   :   8e110014;
        1   :   8e120015;
        2   :   8e130016;
        3   :   8e140017;
        4   :   8e150018;
        5   :   02349022;
        6   :   0255a020;
        7   :   8e320014;
        8   :   0255a020;
        9   :   12330003;
        a   :   00000020;
        b   :   00000020;
        c   :   00000020;
        d   :   ae740032;
        e   :   8e710032;
        f   :   00000020;
END;
```

(3) 时序仿真

时钟周期为 80ns，所以共需要 1.45μs 来执行这 16 条指令




通过波形图可以看出，这个流水线 CPU 可以正常工作。所有的运算结果都正确，

所有的数据相关问题和控制相关问题都被成功解决了。

## 4. 效果评价

我设计的处理器包括了一个 CPU 的所有基本部件，可以分 5 个阶段执行一条 MIPS 指令。因为时间紧张，我只能使这个 CPU 处理 5 种 MIPS 指令，并且 ALU 不能进行乘除运算。但在我设计的这个架构上，可以较容易地扩展出这些功能。

另外，我的 CPU 可以避免由数据相关和控制相关问题引发的错误，从而解决了困扰流水线 CPU 设计的主要问题。定向传送单元虽然简单，但是十分有效。如果有充足的时间，这个 CPU 还可以被扩展出处理中断的功能。流水线技术改善了执行指令的平均时间。我的单周期 CPU 的时钟周期是 160ns，而流水 CPU 的时钟周期只有 80ns。当然，流水 CPU 的性能还可以通过更高级的流水技术来改善。

自己设计一个 CPU 是一个复杂而且耗时的过程，这需要严谨的思考和仔细的设计。尽管在这个过程中我遇到了很多意想不到的困难和障碍，但我还是在两周的时间内完成了这个 CPU 的设计。这次设计的过程增强了我对计算机组成原理的理解，并且会成为我一段美好的回忆。

我的设计方法主要参考了 Patterson 和 Hennessey 所著的《计算机组成与设计》。可以说，这个处理器反映了我对这本教材的内容的理解。在设计过程中，我尽量使用与书中相同的器件名和端口名，以此希望我的 CPU 可以作为以后使用这本书为教材的同学的参考。

# 附录 A: VHDL code for control unit

```
--Control Unit
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY CtrlUnit IS
PORT
(
 instOP              : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
 instFUNC       : IN STD_LOGIC_VECTOR (5 DOWNTO 0);

 ALUOP               : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
 BRANCH              : OUT STD_LOGIC;
 REGWRITE            : OUT STD_LOGIC;
 REGDST              : OUT STD_LOGIC;
 MEMWRITE            : OUT STD_LOGIC;
 MEMTOREG            : OUT STD_LOGIC;
 ALUSRC              : OUT STD_LOGIC
);
END ENTITY CtrlUnit;

ARCHITECTURE SelectResult OF CtrlUnit IS
BEGIN
 ALUOP <=      "1101" WHEN (
                        instOP = "000000" and instFUNC = "100010"--A 减 B，有符号
              ) else

 "0110" WHEN (
                        (instOP = "000000" and instFUNC = "100000")
 or     (instOP = "101011" ) -- SW
                    or     (instOP = "100011" ) -- LW

              ) else
 "1111"; --Not defined yet


 BRANCH<= '1' WHEN (
                    (instOP = "000100") -- B,BEQ
        ) else
 '0';

 REGWRITE <= '1' WHEN (
                    (instOP = "000000"  ) -- Special
```

```
            or    (instOP = "100011" ) -- LW
        ) else
    '0';

    REGDST <= '1' WHEN (
                (instOP = "000000" and instFUNC = "100000") -- ADD
            or    (instOP = "000000" and instFUNC = "100010") -- SUB

        ) else
    '0';

    MEMWRITE <= '1' WHEN (
                (instOP = "101011" ) -- SW
        ) else
    '0';

    MEMTOREG <= '1' WHEN (
                (instOP = "100011" ) -- LW
        ) else
    '0';

    ALUSRC <= '1' WHEN (
                (instOP = "100011" ) -- LW
            or    (instOP = "101011" ) -- SW
        ) else
    '0';


END ARCHITECTURE SelectResult;
```

# 附录 B: VHDL code for Forward Logic

```
--Forward Logic
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ForwardLogic IS
 PORT
 (
        EX_REGWRITE          : IN STD_LOGIC;
        MEM_REGWRITE : IN STD_LOGIC;
        EX_MEMTOREG          : IN STD_LOGIC;
        ID_MEMTOREG          : IN STD_LOGIC;
```

```
        ID_RS               : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        ID_RT               : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        EX_RD               : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        IF_RS               : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        IF_RT               : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        MEM_RD                  : IN STD_LOGIC_VECTOR (4 DOWNTO 0);

        FORWARDA        : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
        FORWARDB        : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
        DELAY               : OUT STD_LOGIC
 );
END ENTITY ForwardLogic;

ARCHITECTURE ForwardResult OF ForwardLogic IS
BEGIN
DELAY <= '1' WHEN (
            (ID_MEMTOREG = '1' and ((ID_RT = IF_RS) or (ID_RT = IF_RT))
        ))
        ELSE '0';

FORWARDA <= "10" WHEN (
            ( EX_REGWRITE = '1' and EX_MEMTOREG = '0'
and ID_RS = EX_RD)
        )
        ELSE      "01" WHEN (
            (MEM_REGWRITE = '1'
and ID_RS = MEM_RD)
        )
        ELSE      "00";

 FORWARDB <= "10" WHEN (
            ( EX_REGWRITE = '1' and EX_MEMTOREG = '0'
and ID_RT = EX_RD)
        )
        ELSE      "01" WHEN (
            (MEM_REGWRITE = '1'
  and ID_RT = MEM_RD)
        )
        ELSE      "00";

END ARCHITECTURE ForwardResult;
```