上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

实验报告

EXPERIMENT REPORT

5 段流水线 CPU 设计与外部 I/O 接口拓展

学生姓名: XXX

学生学号: XXX

班　　级: XXX

目录

实验目的

1. 理解计算机指令流水线的协调工作原理，初步掌握流水线的设计和实现原理。
2. 深刻理解流水线寄存器在流水线实现中所起的重要作用。
3. 理解和掌握流水段的划分、设计原理及其实现方法原理。
4. 掌握运算器、寄存器堆、存储器、控制器在流水工作方式下，有别于实验二的设计和实现方法。
5. 掌握流水方式下，通过 I/O 端口与外部设备进行信息交互的方法。

实验仪器

1. DE1-SOC Cyclone V 实验板 1 套
2. Altera Quartus II 13.1 软件
3. ModelSim-Altera 10.1d 软件

实验任务

1. 采用 Verilog 在 quartus II 中实现基本的具有 20 条 MIPS 指令的 5 段流水 CPU 设计。
2. 利用实验提供的标准测试程序代码，完成仿真测试。
3. 采用 I/O 统一编址方式，即将输入输出的 I/O 地址空间，作为数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址。
4. 利用设计的 I/O 端口，通过 lw 指令，输入 DE2 实验板上的按键等输入设备信息。即将外部设备状态，读到 CPU 内部寄存器。
5. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE2 实验板上的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（或可直接连接至外部设备的控制输入信号）。
6. 利用自己编写的程序代码，在自己设计的 CPU 上，实现对板载输入开关或按键的状态输入，并将判别或处理结果，利用板载 LED 灯或 7 段 LED 数码管显示出来。
7. 例如，将一路 4bit 二进制输入与另一路 4bit 二进制输入相加，利用两组分别 2 个 LED 数码管以 10 进制形式显示“被加数”和“加数”，另外一组 LED 数码管以 10 进制形式显示“和”等。（具体任务形式不做严格规定，同学可自由创意）。
8. 在实现 MIPS 基本 20 条指令的基础上，掌握新指令的扩展方法。
9. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供以上两种指令集（MIPS 和 Y86）应用功能的程序设计代码，并提供程序主要流程图。

实验过程

总体思路

本次实验的设计主要由两部分组成：5 段流水线 CPU 设计与外部 IO 拓展。第一部分，5 段流水线 CPU 设计，主要参考课件中给出的 CPU 设计图（见图 1）与实验手册中给出的代码框架，依次完成 5 段流水线中的各个阶段与段间寄存器的设计；第二部分，外部 IO 拓展，可采用与单周期 CPU 中完全一致的实现方法，故本报告中不加以赘述。

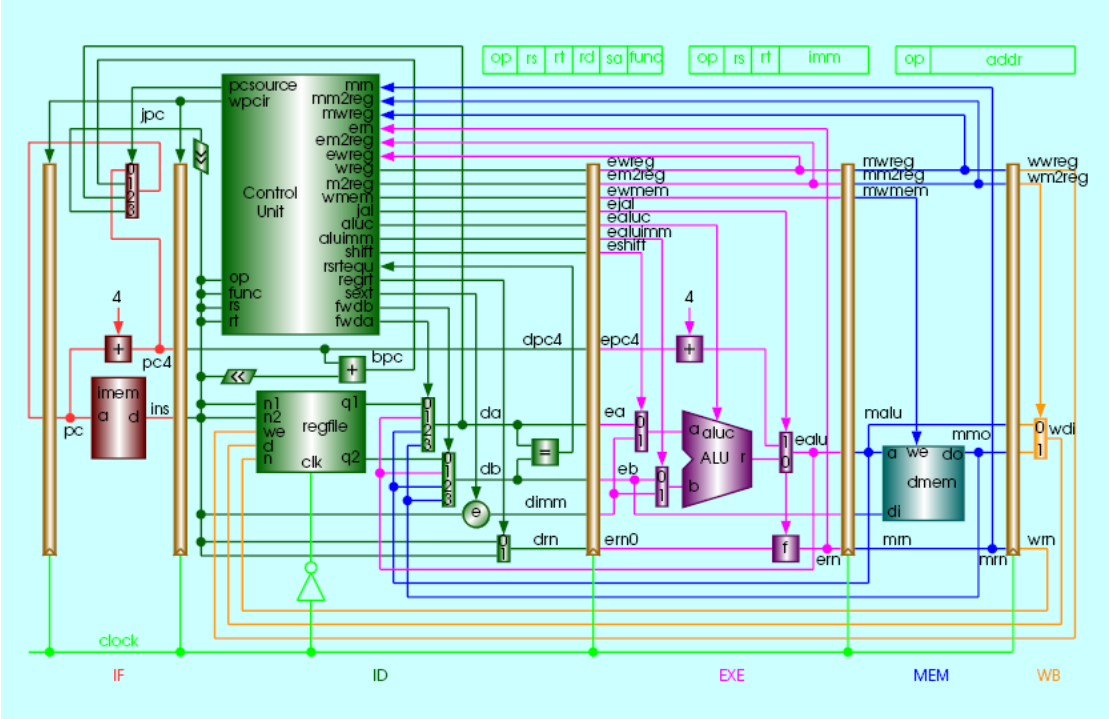

图 1：5 段流水线 CPU 设计

据此，在实验手册中给出的代码框架基础上，稍加修改，得到本实验 Verilog 代码的总体框架，代码如下：（仅 pipeline_computer.v 文件的主体部分，略去部分代码）

```
PC prog_cnt (npc, wpcir, clock, resetn, pc);
IF_stage if_stage (pcsource, pc, bpc, da, jpc, mem_clock, npc, pc4, ins);
IF_ID_register inst_reg (pc4, ins, wpcir, clock, resetn, dpc4, inst);
ID_stage id_stage (
    mwreg, mrn, ern, ewreg, em2reg, mm2reg, dpc4, inst,
    wrn, wdi, ealu, malu, mmo, wwreg, clock, resetn,
    bpc, jpc, pcsource, wpcir, dwreg, dm2reg, dwmem, daluc,
    daluimm, da, db, dimm, drn, dshift, djal, dsa
);
```

```
ID_EXE_register de_reg (
    dwreg,dm2reg,dwmem,daluc,daluimm,da,db,dimm,drn,dshift,dsa,
    djal,dpc4,clock,resetn,ewreg,em2reg,ewmem,ealuc,ealuimm,
    ea,eb,eimm,ern0,eshift,esa,ejal,epc4
);
EXE_stage exe_stage (ea,eb,eimm,epc4,esa,ern0,ealuc,ealuimm,
    eshift,ejal,ealu,ern);
EXE_MEM_register em_reg (
    ealu, eb, ern, ewreg, em2reg, ewmem, clock, resetn,
    malu, mb, mrn, mwreg, mm2reg, mwmem
);
MEM_stage mem_stage (malu, mb, mwmem, clock, mem_clock, resetn,
    mmo,in_port0,in_port1,in_port2,
    out_port0, out_port1, out_port2);
MEM_WB_register mw_reg (
    malu, mmo, mrn, mwreg, mm2reg, clock, resetn,
    walu, wmo, wrn, wwreg, wm2reg
);
WB_stage wb_stage (walu,wmo,wm2reg,wdi);
```

在完成了各个组成部分的设计后，本次实验对应的 Quartus 项目的 Top-Level Entity 的最终结构如图 2 所示:

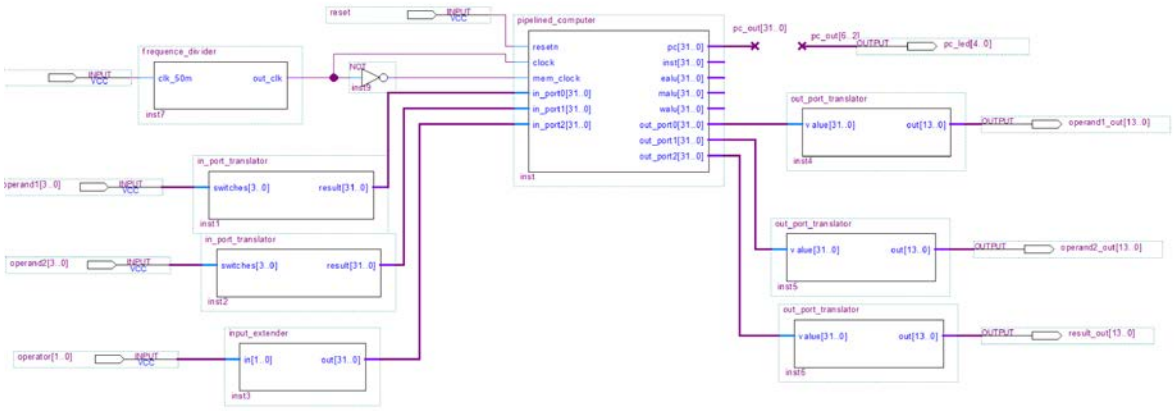

图 2:Top-Level Entity

流水线段间寄存器设计

包括 PC 在内，5 段流水线 CPU 中总共有 5 个段间寄存器需要设计。他们之间具有很大

的相似性：都类似于 D 触发器，在时钟上升沿进行数据写入。其差异在于：若遇到无法解决的数据冒险，流水线不得不停顿一个周期时，PC 和 IF/ID 间寄存器需根据 wpcir 信号进行停顿。以 IF/ID 间寄存器为例（其他寄存器除输入信号的数量不同外，实现上类似），对应的 Verilog 代码如下：

```
module IF_ID_register (
    input [31:0] pc4, ins,
    input wpcir, clock, reset,
    output reg [31:0] dpc4, inst
);
    always @(posedge clock) begin
        if(~reset) begin
            dpc4 <= 0;
            inst <= 0;
        end else begin
            if(wpcir) begin
                dpc4 <= pc4;
                inst <= ins;
            end
        end
    end
endmodule
```

流水线阶段设计（以 ID 阶段为例）

本次实验中流水线 CPU 包含 5 个阶段：IF，ID，EXE，MEM，WB。每个阶段所实现的功能主要参照图 1 的设计图完成，代码较为冗长，在此不便完整贴出，可以访问我的 Github 仓库[1]查看。在此，仅以 ID 阶段为例，对我的设计加以阐述。

ID 阶段的设计，主要包括：指令译码，访问寄存器堆，旁路转发等。接下来分别对这些功能的实现进行介绍。

首先是指令译码，这一部分主要通过控制单元 CU 实现，而本实验中的 CU 是以单周期 CPU 中的 CU 为基础（CU_part）完成的。通过 CU_part 生成初步的控制信号，然后再额外生成流水线控制所需的其他信号（比如流水线停顿信号），若检测到需要停顿流水线，则无视 CU_part 的输出而将所有信号置零。CU 模块的部分 Verilog 代码如下：

[1] Github 仓库的地址：https://github.com/keithnull/Learning-EI332

```
CU_part cu_part(op, func, rsrtequ, dwmem_temp, dwreg_temp, regrt,
     dm2reg_temp, daluc_temp, dshift_temp, daluimm_temp, pcsource,
     djal_temp, sext );
// if wpcir == 0, then all these xxx_temp will be discarded
// if wpcir == 1, then all these xxx_temp will be adopted
assign  wpcir = ~(em2reg & ((ern == rs)|(ern == rt)) & ~dwmem_temp);
assign  dwreg = wpcir?dwreg_temp:1'b0;
assign  dm2reg = wpcir?dm2reg_temp:1'b0;
assign  dwmem = wpcir?dwmem_temp:1'b0;
assign  daluimm = wpcir?daluimm_temp:1'b0;
assign  dshift = wpcir?dshift_temp:1'b0;
assign  djal = wpcir?djal_temp:1'b0;
assign  daluc = wpcir?daluc_temp:4'b0;
```

然后是寄存器堆的访问。为了确保写入时数据已经稳定可用，设定寄存器堆在时钟**下降沿**到来时检查是否需要写入，从而给数据和信号半个周期的传递时间。若非如此，则会导致实际运行时写入数据的不稳定，出现异常情况。寄存器堆的实现基本和单周期 CPU 时基本一致，较为简单，故在此略去代码。

至于旁路转发，在 CU 中，根据相关输入信号进行判断，生成不同转发情况对应的控制信号，然后只需要根据转发控制信号，利用多路选择开关实现即可。如下:

```
// forwarding signals generated in CU
assign fwda[0] = (ewreg & ~em2reg & ern == rs & ern != 0)
    | (mm2reg & mrn == rs & mrn != 0);
assign fwda[1] = (mwreg & ~mm2reg & mrn == rs & ern != rs & mrn != 0)
    | (mm2reg & mrn == rs & mrn != 0);
assign fwdb[0] = (ewreg & ~em2reg & ern == rt & ern != 0)
    | ( mm2reg & mrn == rt & mrn != 0);
assign fwdb[1] = (mwreg & ~mm2reg & mrn == rt & ern != rt & mrn != 0)
    | (mm2reg & mrn == rt & mrn != 0);

// forwarding units in ID_stage
mux4x32 forwarding_da(q1,ealu,malu,mmo,fwda,da);
mux4x32 forwarding_db(q2,ealu,malu,mmo,fwdb,db);
```

关于流水线冒险

考虑流水线可能遇到的三种冒险：结构冒险，数据冒险，控制冒险。

结构冒险，在本次实验的 5 段流水线 CPU 设计中，将存储器分为指令存储器和数据存储器，同时寄存器堆支持在一个周期内同时写入与读取，故不存在结构冒险的可能性。

数据冒险，通过旁路转发的方式，可以解决大多数情形下的数据冒险，唯一的例外是加载-使用型数据冒险（即一条指令需要用到上一条 lw 指令的结果）。为了解决这个问题，在本次实验的设计中，利用冒险检测单元，生成 wpcir 信号，从而在遇到此情形时让流水线停顿一个周期。

控制冒险，本次实验所设计的 CPU 中 PC 的可能来源有 4 个，则带来潜在的控制冒险的风险。为了解决此问题，简单起见，可采用转移延迟时间片（branch delay slot）的方式。

I/O 拓展

同单周期 CPU 实验类似，本次实验的 I/O 拓展采用了基本相同的方法，故不赘述。

编写程序

同单周期 CPU 实验类似，本次实验中所编写的汇编程序基本相同，故不赘述。

实验总结

相比于单周期 CPU 实验给出了大部分的 Verilog 代码，本次实验仅仅提供了顶层设计框架供参考，因而实验过程花了更多的时间与心思。参照着 5 段流水线 CPU 的设计图，我实现了流水线的各个阶段以及段间寄存器，这一过程总的来说比较顺利，但是也遇到了一些问题：比如 CU 中纷繁的控制信号生成，转发单元的设计等等。解决这些问题，让我较为深入地理解了 5 段流水线 CPU 的运行方式。