实验目的：

1. 理解计算机的外部 I/O 扩展方法原理。

2. 理解外部 I/O 总线的设计原理。

3. 理解 I/O 地址空间的设计方法，掌握 I/O 端口的设计方法。

4. 会通过设计 I/O 端口与外部设备进行信息交互。

实验内容：

1. 在实验二实现的类 MIPS 指令系统的 5 段流水 CPU 设计基础上，采用 I/O 地址和主存统一编址的方式，即将输入输出的 I/O 地址空间，作为主存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。

2. 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。即将外部设备的状态或数据输入，读入到 CPU 内部寄存器。3. 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（该控制寄存器输出，可直接连接输出至外部设备，作为对外部设备的控制输入信号）。

4. 能够利用自己编写的程序代码，在自己设计的 CPU 上，利用 DE2 板载资源，设计实现一个简单的计算器。

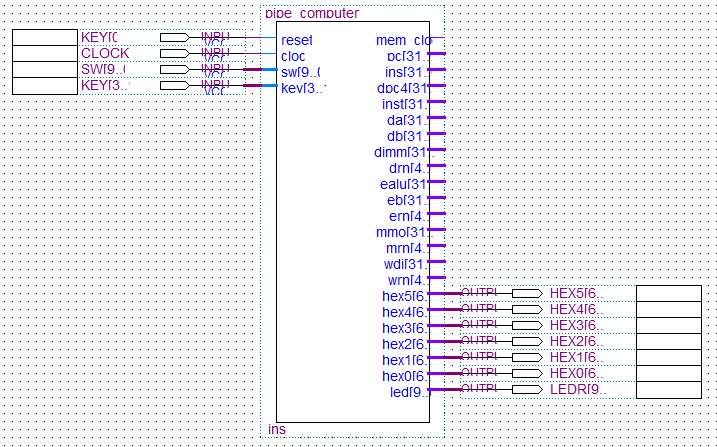
5. 利用板载的 VGA 接口资源，设计显示缓存 Framebuffer 存储器、及相应的显示控制逻辑，实现计算结果或其它内容在 VGA 接口显示器上的显示。（扩展内容）

6. 也可以基于板载资源，进行其它多种接口控制器（如 RS232 串行接口、USB接口、以太网接口等）通过 I/O 总线的互联控制。（扩展内容）

7. 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，提供以上应用功能的程序设计代码，并提供程序主要流程图。

实验设计：

顶层设计：



主模块：

具体代码已由实验指导书给出。该计算机的工作被划分为 5 个阶段（取指、译码、执行、访存、写回），每个阶段前有一个流水线寄存器用于锁存阶段之间需要传递的信息，形成 5 段流水线。

代码实现：

IF 流水段前寄存器

此寄存器锁存 PC 值信号。 clock 、 resetn 、 wpcir 分别作为时钟信号、重置信号和（写入）使能信号。使能信号用于实现暂停（stall）。这里使用了一个特殊的 D 触发器（ dffe32pc ），它会在重置时把值置为 -4，以保证第一条指令（PC = 0）能正常执行。

取指阶段

取指阶段根据译码阶段传来的 pcsource 信号及各个可能的新 PC 位置计算下一条指令取指的地址，同时访问指令存储取出当前指令。指令存储模块（ pipe\_instmem ）与单周期的设计基本相同，唯一的区别在于时钟信号上。这里采用了 rom\_clock 信号为其时钟信号（同步读），它与 clock 信号反相，也即在每个周期一半的时刻读出指令（预留半个周期的时间等待信号稳定）。

IF/ID 流水线寄存器

此寄存器锁存 IF 和 ID 阶段需要传递的信号。 clock 、 resetn 、 wpcir 分别作为时钟信号、重置信号和（写入）使能信号。使能信号用于实现暂停（stall）。32 位 D 触发器（ dffe32 ）在重置时将值清零，其余行为与前述的 dffe32pc 相同。

译码阶段

译码阶段模块包括控制器（ pipe\_cu ）、寄存器文件（ regfile ）、几个多路选择器以及一些链路等。各类指令的处理方法与单周期基本相同。以下是几个不同之处：

译码阶段提前计算的 rsrtequ 信号取代了原来单周期由 ALU 产生的 z 条件码传递给控制器。

sa 和 imm 两个信号被合并为一个信号向后传输（它们不会同时出现）。

控制器需要检查是否出现流水线冒险情况，并做出相应的处理。

寄存器文件改为在 clock 下降沿写入（即在周期的一半时刻写入，方便读出同一周期内写回阶段写入的值），其余行为与单周期相同。

selecta 和 selectb 两个多路选择器按照控制器传来的选择信号，从寄存器读取结果和几个转发源中选择应当使用的值。

控制器产生的大部分控制信号与原来单周期的相同，新增部分用于处理流水线冒险情形。

数据冒险：

大部分数据冒险都能用转发来解决。转发源可以是上一条指令（当前 E阶段的 ALU 结果）或上上条指令（当前 M 阶段的ALU 结果、读内存结果）。需要考虑到转发的优先级（由近至远），同时禁止转发 0 号寄存器（因为对 0 号寄存器的写入是无效的）。

加载/使用冒险不能用转发来解决，因为必须等待一个周期才能获得内存中的数据。当检测到这种冒险情形时，控制器向F 和 D 寄存器发出暂停信号（ wpcir 置为 0），同时向 E寄存器传递一个气泡（将 wreg 和 wmem 信号置为 0，禁止写入寄存器和存储器，即相当于一条 nop 指令）。

控制冒险：本 MIPS 处理器使用延迟槽（delay slot）特性消除了控制冒险。对于任意的转移指令（无条件跳转、条件跳转、调用和返回），其下一条指令（PC + 4）都会在转移指令之后被执行，然后再采取转移行动。第 N 条指令应该从哪里取是由第N - 2 条指令决定，而非由第 N - 1 条指令决定。这样的特性简化了流水线的设计，因为不需要实现分支预测逻辑了（也不会再有分支预测错误的处罚），译码阶段一定能得到正确的转移目标。缺点是这样指令集的语义会有点奇怪。

ID/EXE 流水线寄存器

此寄存器锁存 ID 和 EXE阶段需要传递的信号。 dff1 、 dff4 、 dff5 、 dff32 为不同位数的 D 触发器，除无使能信号以外，其余行为与前述的 dffe32 相同。

执行阶段

执行阶段与原单周期的对应部分逻辑基本相同。不同之处在于：对 jal 指令的处理被放进了这一阶段。由于延迟槽特性，写入的返回地址应由 PC + 4 改为 PC + 8。ALU 除不再产生 z 条件码以外，其余行为与原来单周期相同。

EXE/MEM 流水线寄存器

此寄存器锁存 EXE和 MEM 阶段需要传递的信号。

访存阶段

访存阶段访问数据存储模块（ pipe\_datamem ）完成内存数据及 I/O 端口的读写。同步读写 RAM 的时钟改用与 clock 信号反相的ram\_clock 信号，也即在每个周期一半的时刻读写内存（预留半个周期的时间等待信号稳定）。其余行为与单周期相同。

MEM/WB 流水线寄存器

此寄存器锁存 MEM 和 WB 阶段需要传递的信号。

写回阶段

写回阶段按照选择信号的指示，从 ALU 结果和读取 RAM 的结果中选取应当写回到寄存器文件的值。使用 CPU 实现计算器

此计算器程序与单周期实验中设计的行为基本相同（能执行加、减、异或三种运算），唯一的不同点在于调整了部分代码以适应延迟槽特性，故不再赘述。

start: j main\_loop

lui $7, 0

sevenseg: sll $30, $30, 2

jr $ra

lw $29, 0($30)

split: add $29, $0, $0

split\_loop: addi $30, $30, -10

sra $28, $30, 31

bne $28, $0, split\_done

add $0, $0, $0

j split\_loop

addi $29, $29, 1

split\_done: jr $ra

addi $28, $30, 10

show: add $20, $31, $0

sll $26, $29, 5

jal split

addi $26, $26, 0xff20

jal sevenseg

add $30, $29, $0

sw $29, 16($26)

jal sevenseg

add $30, $28, $0

sw $29, 0($26)

add $31, $20, $0

jr $ra

get\_op: lw $5, 65296($0)

addi $6, $0, -1

xor $5, $5, $6

andi $6, $5, 0x8

bne $6, $0, add\_op

add $0, $0, $0

andi $6, $5, 0x4

bne $6, $0, sub\_op

add $0, $0, $0

andi $6, $5, 0x2

bne $6, $0, xor\_op

add $0, $0, $0

jr $ra

add\_op: addi $6, $6, -5

sub\_op: addi $6, $6, -3

xor\_op: jr $ra

add $7, $6, $0

do\_op: bne $7, $0, not\_add

add $0, $0, $0

jr $ra

add $4, $2, $3

not\_add: addi $8, $7, -1

bne $8, $0, not\_sub

sub $4, $2, $3

sra $5, $4, 31

beq $5, $0, sub\_done

add $0, $0, $0

sub $4, $0, $4

sub\_done: jr $ra

add $0, $0, $0

not\_sub: jr $ra

xor $4, $2, $3

main\_loop: lw $1, 65280($0)

sw $1, 65408($0)

andi $2, $1, 0x3e0

jal get\_op

srl $2, $1, 5

jal do\_op

andi $3, $1, 0x1f

add $30, $4, $0

jal show

addi $29, $0, 0

add $30, $2, $0

jal show

addi $29, $0, 2

add $30, $3, $0

jal show

addi $29, $0, 1

j main\_loop

add $0, $0, $0 仿真验证

`timescale 1ps/1ps

module pipe\_computer\_sim;

reg resetn, clock;

wire mem\_clock;

wire [31:0] pc, ins, dpc4, inst, da, db, dimm, ealu, eb, mmo, wdi;

wire [4:0] drn, ern, mrn, wrn;

reg [9:0] sw;

reg [3:1] key;

wire [6:0] hex5, hex4, hex3, hex2, hex1, hex0;

wire [9:0] led;

pipe\_computer\_main pipe\_computer\_instance(resetn, clock, mem\_clock,pc, ins, dpc4, inst, da, db, dimm, drn, ealu, eb, ern, mmo, mrn, wdi, wrn,sw, key, hex5, hex4, hex3, hex2, hex1, hex0, led);

initial begin // Generate clock.

clock = 1;

while (1)

#2 clock = ~clock;

end

initial begin // Generate a reset signal at the start.

resetn = 1;

#1 resetn = 0;

#5 resetn = 1;

// while (1) begin // Reset and run pipe test again.

// #400 resetn = 0;

// #5 resetn = 1;

// end

end

initial begin // Simulate switch changes.

sw <= 10'b1010101010;

while (1)

#2400 sw = ~sw;

end

initial begin // Simulate key presses.

key <= 3'b111;

while (1) begin

#800 key <= 3'b101; // key2 pressed, should change to sub mode

#800 key <= 3'b110; // key1 pressed, should change to xor mode

#800 key <= 3'b011; // key3 pressed, should change to add mode

end

end

endmodule

该仿真测试代码模拟出 CPU 工作所需要的时钟信号和复位信号，此外在运行不同汇编程序时有不同的仿真功能：

在运行流水线测试程序时，可通过周期性的 reset 来反复运行测试代码验证流水线 CPU 的正确性。（通过取消注释上面的 4行代码开启）

在运行计算器程序时，可在适当的时刻模拟开关和按钮的输入。

可在仿真过程中观察各个输出信号的值是否正确，以验证流水线 CPU 工作是否正常。

实验心得：

在本次实验中我实现了一个 5 段流水线 CPU，并在其上用汇编码实现出了一个 简单的多功能计算器，更清晰深刻地理解了流水线各个阶段之间的协调配合以及各种流水线冒险的处理方式。本次实验的代码相对上次试验而言主要增加了关于五个步骤的10个module，其中，比较复杂的是数据冒险部分，但通过课上以及课后的学习，我顺利完成了这部分代码的编写。

同时，本次实验也让我了解到了下手写代码之前应该仔细阅读和思考整体的架构，充分理解每一处细节，然后再开始会顺利很多。同时，遇到问题不能心急，想出有效的办法定位问题之所在才是关键，必要时和老师、同学沟通能更有效地解决问题。