2019 Spring ICSII

计算机体系结构实验

实验二

32位MIPS多周期处理器

实验报告

姓名：田睿

学号17300750084

CONTENT

1 实验目的

熟悉单周期MIPS处理器的工作原理，掌握使用verilog设计硬件处理器的方法；

①完成MIPS单周期处理器设计。包含十四条基本指令；

②编写MIPS汇编测试代码测试MIPS处理器。

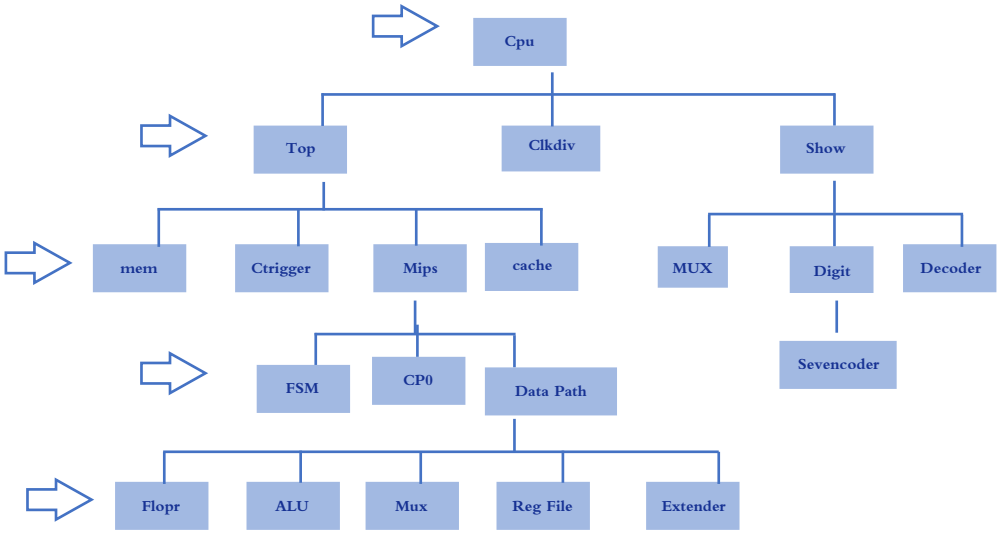
③在NEXYS4 DDR板上进行验证。

2 32位MIPS多周期处理器313的设计思路

2.1 处理器的总体结构

与单周期相似，在多周期的处理器中我使用模块例化的方法，将CPU拆解为多个模块：其中我将显示模块与CPU核心模块分开，显示模块主要包括了七段数码管和LED的展示实现；CPU的核心模块则主要有控制器（有限状态机）、数据通路以及存储器实现

多周期的层级结构及模块关系如下图所示：



2.2 顶层模块-Cpu_show模块 [1]

Cpu_show作为顶层模块，链接内部核心与外部显示端口。输入输出信号全部来自或输出至NEXYS DDR板。输入CLK100MHZ的外部时钟信号，以及NEXYS DDR板的开关决定的控制信号，以及由BTNC的reset信号，输出七段数码管的使能信号AN、显示信号CA以及

在cpushow的模块下有由clkdiv、mips_top、show三个模块实例化的组件；

2.3 分频器 clkdiv

实现细节见单周期 2.3

2.4 显示模块 show

show 模块是输出主要显示信号的枢纽，其输入信号有：clkinfo、instr、memdata、pcinfo、regdata、stateinfo、cachedata 、 causeinfo 这八个信号分别为时钟信号（相比单周期的show模块，增加了stateinfo、cachedata以及causeinfo）；它们将作为data_mux的输入信号，由开关控制决定七段数码管显示的数据；

输入两个16-bit信号：sign_one、sign_two为 led显示提供驱动信号；

同时输入cache_rtg 、cache_rs 两个2位信号，用以显示显示的cache数据的组号以及组内偏移量；

输入控制信号sw则有多个用途：sw[15:13]则作为data_mux的选择信号，sw[12:5]是decoder的输入编码；经由decoder译码得出的内存以及寄存器地址选择信号mem_s、reg_s输出，同时与NEXYS DDR板相关联的CA、En、LED信号也是输出信号；sw[15:13] 以及 sw[3:2]、sw[1]将共同决定led管输出的信号

show模块下有mux8_32、decoder、digit_show、cache_show、mux2_16、led_s 五个模块实例化的组件；

与单周期相似：我的展示模块具有很好的友好性。使用者可以便捷地通过切换开关看到mips处理器内部的数据：

其中 mux8_32 将外部的开关信号作为多选器的选择信号选择内部数码管输出的数据信息，对应如下：

sw[15:12]	3’b00	3’b001	3’b010	3’b011	3’b100	3’b101	3’b110	3’b111
显示数据	pc值	指令	时钟周期数	状态机状态	cache数据	通用寄存器数据	内存数据	cause寄存器数据

相应的NEXYS DDR板示意图如下：

与此同时，sw[1]开关则控制led段接收到的sign信号，

LED	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sw[1] = 0	Ext op	IntCause	CauseWrite	EPCWrite	Update	PCEn	WriteBack	IorD	Regwrite	IRWrite	MemWrite	Dirty	Miss	Overflow	Negative	Zero
sw[1] = 1		ALUcontrol				PCSrc			Mem2Reg		RegDst		ALUSrcA		ALUSrcB	

2.5 控制信号处理模块 FSM

FSM 模块是mips多周期处理器重要的信号处理器件。只有当有限状态机输出正确状态以及转换时，多周期处理器才可能正常运转。

而多周期的处理器的信号处理则依赖于状态的转换，我设立了23个运行状态：

状态转移图如下

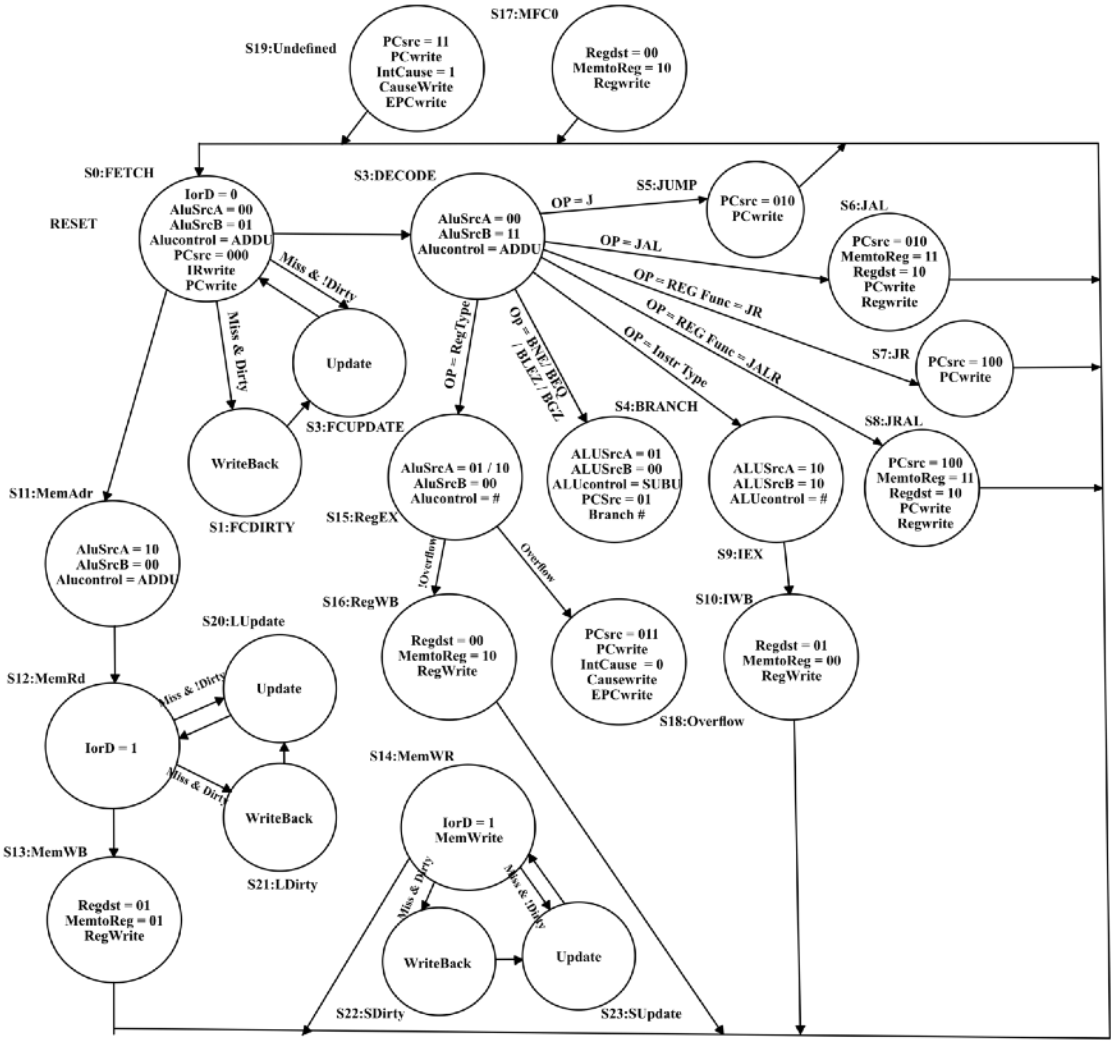


图2.5.1状态转移图（绘制：田睿）

2.6 数据通路模块 datapath

Datapath模块为mips处理器的数据通路；控制信号由control unit输出后进入datapath，由nexys ddr板开关控制的信号也传入数据通路。每条指令的运算与寄存器存取操作都在数据通路中完成。其下有模块：mux2_5、mux2_32、jumpadd、regfile、alu、flopr、extender、pcbranch、reg32_en、reg32等实例化的部件。

1）flopr

实现细节见单周期 2.7（1）

2）pcplus4

实现细节见单周期 2.7（2）

3）pcbranch

实现细节见单周期 2.7（3）

4）jumpadd

实现细节见单周期 2.7（4）

5）寄存器文件 regfile

实现细节见单周期 2.7（5）

6）多用选择器组

在datapath的模块下共有六个多选器，以下为详细阐述。

第一个多选器 iord_mux 为32位2-1多路选择器，其选择信号为Iord，选择输入到缓存地址端的地址信息来自立即数还是PC寄存器的输出端。

第二个多选器 regdst_mux 为5位3-1多路选择器，多选器输出值即为通用选择器写入的地址，当选择信号为00时输出instr[20:16]，为01时输出instr[15:11]，为10时输出32（当遇到jal指令时生效）。

第三个多选器 regwb_mux 为32位4-1多路选择器。选择写入寄存器的数据，片选信号为MemtoReg，当选择信号为00时输出alu结果寄存器的输出值，为01时输出数据寄存器的输出值，为10时输出PC值（当遇到jal指令时生效），当为11时输出C0寄存器的输出值（MFC0指令时生效）。

第四个多选器 srcA_mux 为32位3-1多路选择器输入信号分别为PC寄存器的输出值（00）、位移指令时指令存储的偏移量（01）、以及dataA寄存器的输出值（10），选择信号为ALUSrcA，输出值将作为ALU A端口的输入数据。

第五个多选器 srcB_mux 为32位3-1多路选择器输入信号分别为dataB 寄存器的输出值（00）、4（计算下一PC时生效）、经过符号扩充的立即数（01）以及 左移两位后的立即数（branch类指令时生效），选择信号为ALUSrcB，输出值将作为ALU B端口的输入数据。

第六个多选器 wdata_mux 为32位5-1多路选择器输入信号分别alu结果寄存器的输入信号（00）、alu寄存器的输出信号（01）、计算后的转移pc（jump类指令）（10）、（const）异常处理指令存储地址（11）以及 寄存器文件A端口的输出值（100），选择信号为PCsrc，输出值将作为pc寄存器的输入信号，即为下一pc值。

7）signext 符号扩展

实现细节见单周期 2.7（9）

8）extender 偏移量扩展

实现细节见单周期 2.7（10）

9）ALU

实现细节见单周期 2.7（11）

2.7 七段数码管显示模块

实现细节见单周期 2.8

2.8 缓存 Cache

在本次实验的多周期处理器中，我添加了一级缓存Cache。 模块例化得到的Cache 有clk 、reset、update、add、wdata、d、we、show_tag、show_s作为输入信号；其中clk为Cache的时钟信号（Cache内部实现为时序电路）；update为FSM的输出信号，当传入cache后，cache将从内存中替换数据完成更新，add为处理器内部需要从缓存中读取数据的地址；wdata为需要写入缓存的数据；d为从内存中取得的更新数据（数据宽度为128bit）；we为写入缓存的使能信号，由FSM输出的memWrite控制；show_tag则来自调节NEXYSDDR板时需要查看的缓存的组号，show_s则代表需要查看的组内偏移量；与此同时，cache输出show_data以显示到数码管，show_dirty则表示查看的数据的dirty属性，data为从缓从中读取的数据；miss_i以及dirty_i为cache传出到FSM的信号，若miss则代表cache需要更新并替换新的数据，这是状态机会转入相应的update状态，并且传回update = 1；若miss的同时dirty信号为1，则代表cache更新的同时，还需要将已经改写的信号写回内存，此时状态机会转入相应的dirty状态，cache传出需要写回的数据（writeback）以及写回地址（wadd）给内存，待内存完成写回后，再进入update的状态；

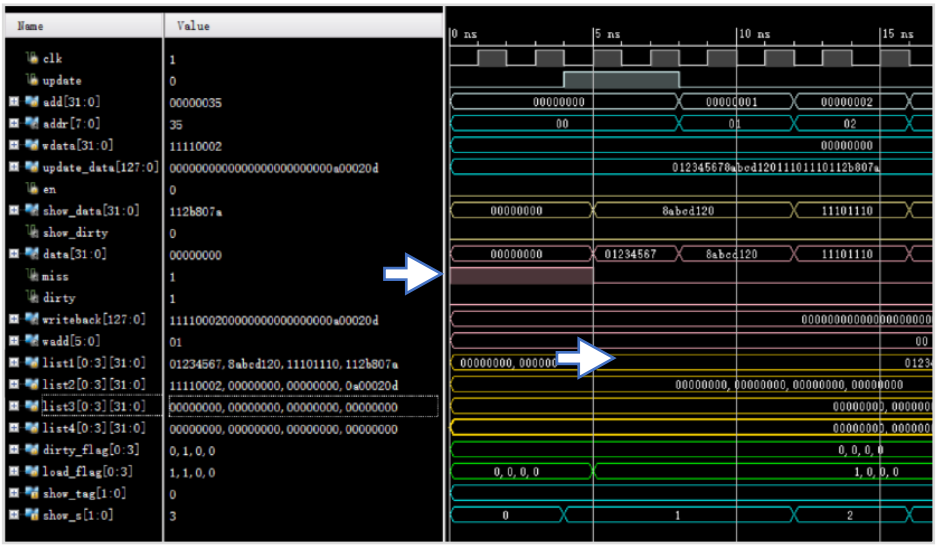
add[7:4]	add[3:2]	add[1:0]					
tag[0]	[00]	[00] list1[0]	[01] list1[1]	[10] list1[2]	[11] list1[3]	Dirty_flag	Load_flag
tag[1]	[01]	[00] list2[0]	[01] list2[1]	[10] list2[2]	[11] list2[3]	Dirty_flag	Load_flag
tag[2]	[10]	[00] list3[0]	[01] list3[1]	[10] list3[2]	[11] list3[3]	Dirty_flag	Load_flag
tag[3]	[11]	[00] list4[0]	[01] list4[1]	[10] list4[2]	[11] list4[3]	Dirty_flag	Load_flag

图2.8.1 Cache 的内部结构

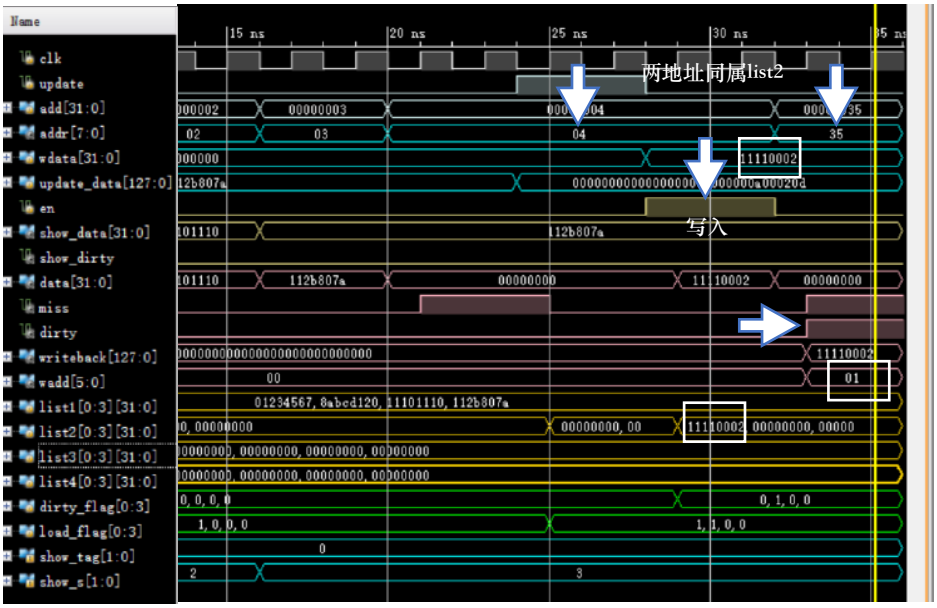
该缓存为 4 * 4 的结构，总共存储 4 * 4 * 32 bit的数据，其中dirty_flag代表每一

组的数据是否被改写过，若被改写则为1，而load_flag则代表数据是否有效，若有效则为1；在初始条件下，cache内部的数据都为0，load_flag默认为0，dirty_flag也默认为0；在数据无效的情况下，cache会直接返回miss的指令。而当数据有效的时候，若传入地址的tag与对应组号的cache数据tag相同，则cache将直接返回读取数据，若不一致，则输出miss信号。当写入使能为1时，若hit，则直接改写cache中数据，并将dirty_flag置1；

2.8.1 针对 cache 的仿真



可见在初始情况下load_flag为1，miss输出信号为1，在update信号为1时，成功将从mem中取得的数据装载；



测试en为1时的写入情况，cache成功写入传入数据到指定地址。当需要读取新的地址信息时，miss与dirty同时变为1；

2.9 内存 Memory

为了减少更新cache所需的周期数，我选择将memory的数据宽度设置为128bit，这样，当需要更换cache的list时，可以直接将4 * 32 bit的数据在一周期内传入cache内。与此同时，memory同样采用了dual - port 的 distributed RAM ， 在本次实验中，我没有区分指令内存以及数据内存。

2.10 CP0模块

该模块并没有实现CP0的常规化全部功能，而是仅仅针对undefined 和 overflow 两个指令做出判断；

CP0中有EPC寄存器、cause寄存器，由于undefined指令以及overflow指令仅仅针对协处理器中的CP0[13]以及CP0[14]，因此EPC寄存器输出的值（出错时的pc值）、原因寄存器的输出值（cause）将在时钟上升沿cp0寄存器；

3 文件清单

- code

Nexys4DDR.xdc

引脚锁定文件

datamem.coe

初始内存的coe代码

FSM.v

包括控制器模块FSM

cache.v

包括缓存数据单元cache；

show.v

mux2_1、led显示模块led_s，译码器decoder，cache显示模块 cache_show，

七段数码管显示模块sevencoder、显示模块digit_show，数据多选器 mux8_32,

cpu.v

包括内存mem、处理器顶层 mips_top、使能时钟ctrigger、处理器核心mips、

异常处理器Cp0、数据通路模块 datapath

datapath.v

包括datapath中所有单元化模块：

reg32_2n、reg32、flopr、mux3_5、mux4_32、regfile、extender、signext

mux3_32、ALU、jump_add、mux5_32

- *simulation*

所有仿真代码以及其他文件存放于以下链接：

https://github.com/Stephyuka/mips_32

4 仿真实验与结果

指令： addi

针对立即数类指令的仿真：可以看见状态跳转为：

decode （03） 》 Iex（09） 》 Iwb（10）

指令： add
针对reg类型指令的仿真：可以看见状态跳转为：

decode（03） 》 regex（15） 》 regwb（16）

指令： beq
针对分支类型指令的仿真：可以看见状态跳转为：

decode（03） 》 branch（04）

在这条指令执行的同时，可见cache更新的状态（upadate为1）：

fetch（00） 》 fcupdate（02）》 fetch（00）

同时可见在update后的周期，list2加载为新的数据；

指令： jump
针对分支类型指令的仿真：可以看见状态跳转为：

decode（03） 》 jump（05）

并且可见在状态变为jump后的一周期，pc值更新为跳转地址

5 申A理由

5.1 处理器操作友好性

- **设计时钟使能信号**

- **设计循环查看开关**
- **支持切换查询不同数据**

详见单周期实验报告

- **设计缓存CACHE**

5.2 指令集及数据通路的扩充

实现指令：ADD \SUB \ADDU\SUBU\AND\OR\XOR\NOR\

SLT \SLTU \ SRL \SRA\ SLL \ SLLV \ SRLV \SRAV \JR \ JALR\

NOP \MFC\ADDI \ANDI \ORI \XORI \ LW \ LUI\ SW \ BEQ\

BNE \BLEZ\BGTZ \SLTI \J \JAL