

**计算机组成原理Project4实验报告**

Verilog – 支持10指令单周期CPU

{addu,subu,ori,lui,beq,sw,lw,jal,jr,nop}

北京航空航天大学

计算机学院

陈麒先

16061160

二○一七年十一月

**郑重声明**

**关于诚实守信公约：**

**本实验报告由本人独立完成，全部内容均为本人通过查找互联网资料、翻阅课件、课堂笔记和教材后独立思考的结果。特此声明。**

**16061160**

**陈麒先**

**原创性声明**

**作业中出现的公式、图片、代码段以及图片的文字注释信息，均为作者原创。抄袭行为在任何情况下都被严格禁止(COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！**

**16061160**

**陈麒先**

**目录**

[**第一章** **设计架构** 2](#_Toc498883720)



[第一节 单周期 CPU 顶层架构视图 2](#_Toc498883721)

[第二节 单周期 CPU 指令数据通路 2](#_Toc498883722)

[第三节 单周期 CPU 模块定义说明 5](#_Toc498883729)

[第四节 单周期 CPU 控制单元设计 9](#_Toc498883735)

[**第二章 测试验证** 11](#_Toc498883736)

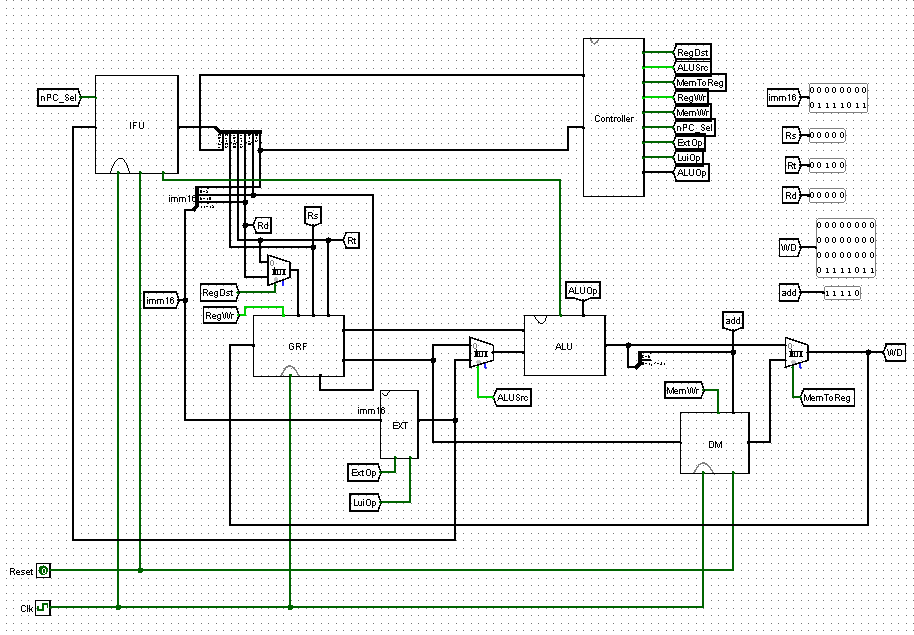
[第一节 测试代码 11](#_Toc498883737)

[第二节 测试期望 14](#_Toc498883741)

[**第三章 课后思考** 18](#_Toc498883745)

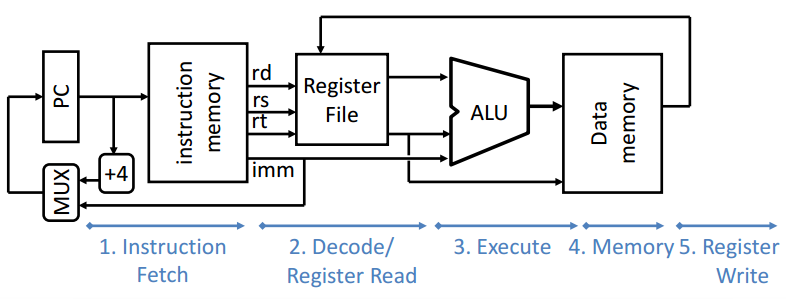
1. **设计架构**

第一节 单周期 CPU 顶层架构视图

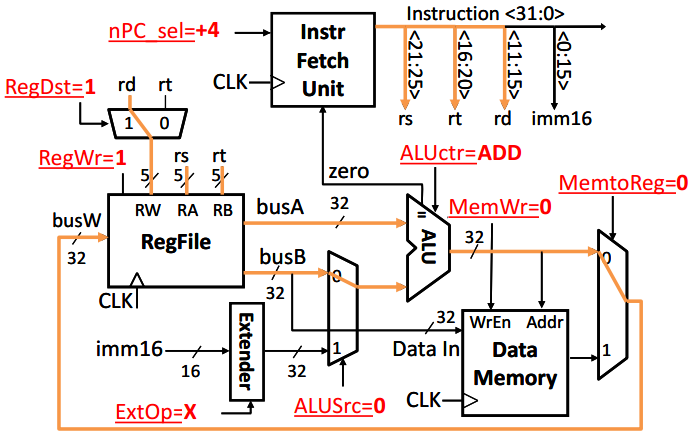


第二节 单周期 CPU 指令数据通路

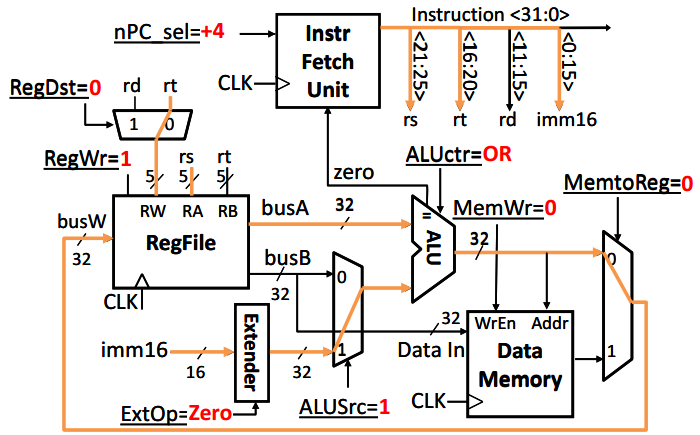
1. 数据通路整体架构



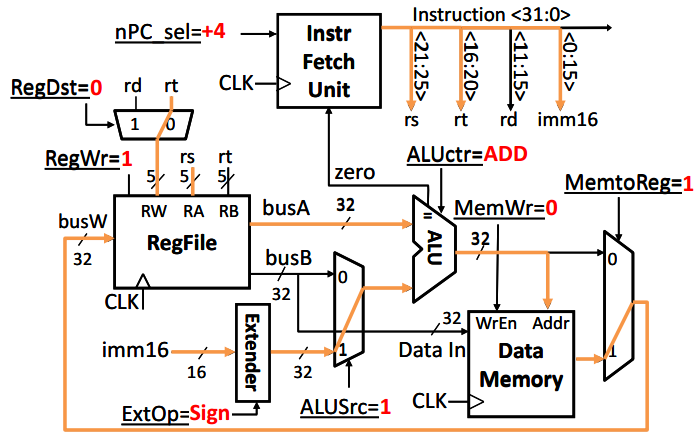
1. addu / subu 指令数据通路



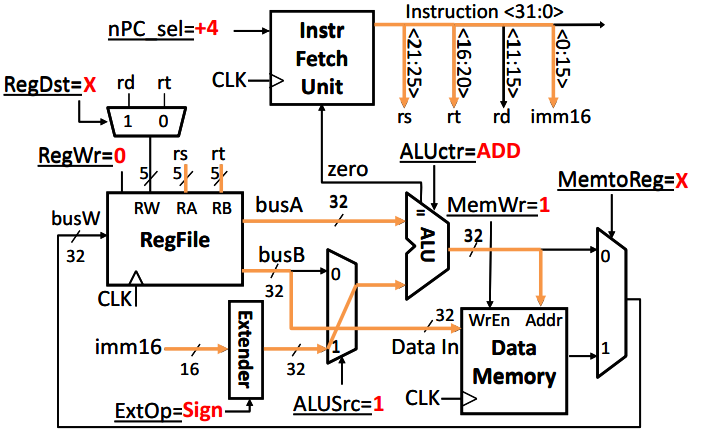
1. ori 指令数据通路



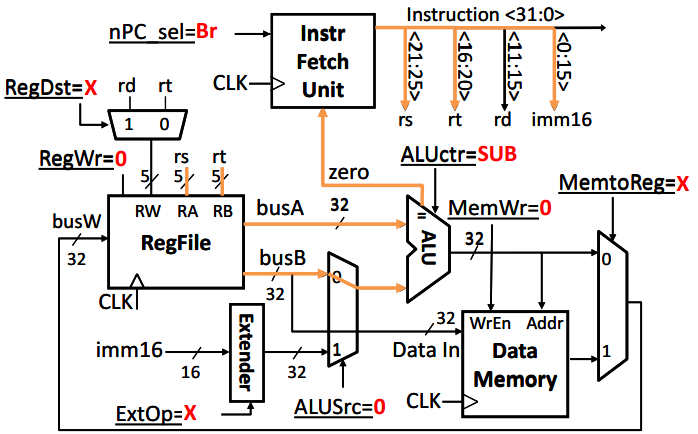
1. lw 指令数据通路



1. sw 指令数据通路



1. beq 指令数据通路



第三节 单周期 CPU 模块定义说明

**1、IFU**

**（1）基本要求**

•起始地址：0x00000000。

•IM用 ROM 实现，容量为 32bit \* 32。

•因 IM 实际地址宽度仅为 5 位，故需要使用恰当的方法将 PC 中储存的地址同 IM 联系起来。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| Reset | In | 异步复位信号 |
| if\_beq | In | 跳转使能信号 |
| zero | In | 相等条件信号 |
| Offset | In | 地址偏移量 |
| Instr | Out | 输出32位指令码 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | Reset信号有效时，PC复位 |
| 2 | 跳转 | nPC\_Sel和zero同时有效时，执行跳转 |
| 3 | 取指令 | PC决定所取指令的地址，每个周期PC+4 |

**2、GPR**

**（1）基本要求**

•用具有写使能的寄存器实现，寄存器总数为 32 个。

•0 号寄存器的值保持为 0。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| Reset | In | 异步复位信号 |
| RA | [4:0] In | 读总线A地址 |
| RB | [4:0] In | 读总线B地址 |
| RW | [4:0] In | 写寄存器地址 |
| WD | [31:0] In | 写数据输入 |
| WE | In | 写使能 |
| BusA | [31:0] Out | 总线A输出 |
| BusB | [31:0] Out | 总线B输出 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 同步复位 | Reset信号有效时，GRF复位 |
| 2 | 读寄存器 | 根据RA，RB所指示的地址，读出对应寄存器的值 |
| 3 | 写寄存器 | 根据RW所指示的地址，将WD的数据写入对应寄存器 |
| 4 | 0号寄存器 | 0号寄存器不连接数据写入端口，输出接地 |

**3、ALU**

**（1）基本要求**

•提供 32 位加、减、或运算及大小比较功能。

•可以不支持溢出（不检测溢出）。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| BusA | [31:0] In | ALU 第一个操作数 |
| BusB | [31:1] In | ALU 第二个操作数 |
| ALUOp | [1:0] In | ALU 控制信号 |
| alu\_output | [31:0] Out | ALU 计算结果 |
| zero | Out | 两个操作数相等比较结果，相等时置1 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 加 | ALUOp = 00,两个操作数相加 |
| 2 | 减 | ALUOp = 01,两个操作数相减 |
| 3 | 或 | ALUOp = 10,两个操作数按位或 |
| 4 | 比较 | 若两个操作数相等，zero置1 |

**4、DM**

**（1）基本要求**

•使用 RAM 实现，容量为 32bit \* 32。

•起始地址：0x00000000。

•RAM 应使用双端口模式。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| Reset | In | 异步复位信号 |
| WE | In | 写使能信号 |
| WD | In | 写入数据 |
| Address | In | 写入地址 |
| RD | Out | 读内存数据 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | Reset信号有效时，DM复位 |
| 2 | 读内存数据 | WE信号为0时，读内存中Address指示的地址数据 |
| 3 | 写内存数据 | WE信号为1时，向内存中Address指示的地址写数据 |

**5、EXT：**

**（1）基本要求**

•可以使用 logisim 内置的 Bit Extender。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| Imm16 | [15:0] In | 输入待扩展的16位立即数 |
| Luiop | In | 位扩展信号 |
| ExtOp | In | 符号扩展信号 |
| Ext32 | [31:0] Out | 扩展结果输出 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 零扩展 | 当LuiOp信号和ExtOp信号均无效时，执行零扩展 |
| 2 | 符号扩展 | 当LuiOp信号无效，ExtOp信号有效时，执行符号扩展 |
| 3 | 位扩展 | 当LuiOp信号有效，ExtOp信号无效时，执行位扩展 |

第四节 单周期 CPU 控制单元设计

**（1）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| OpCode | [5:0] In | 指令高六位 Ins [31:26] |
| Func | [5:1] In | 指令低六位 Ins [5:0] |
| RegDst | Out | 若为高电平，选择Rd，否则选择Rt作为GPR的RW输入 |
| ALUSrc | Out | 若为高电平，选择扩展数字，否则选择BusB作为ALU的第二位输入 |
| MemToReg | Out | 若为高电平，选择DM，否则选择ALUOut作为GPR的WD输入 |
| RegWrite | Out | 若为高电平，则对GPR进行写操作，否则写使能无效 |
| MemWrite | Out | 若为高电平，则对DM进行写操作，否则对DM进行读操作 |
| If\_beq | Out | 若为高电平且zero为高电平，则跳转 |
| ExtOp | Out | 若为高电平，则符号扩展 |
| Luiop | Out | 若为高电平，则位扩展 |
| ALUOp[0] | Out | 共同决定ALU的行为 |
| ALUOp[1] | Out |

**（2）控制器真值表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | ori | lui | lw | sw | beq |
| OpCode | 000000 | 000000 | 001101 | 001111 | 100011 | 101011 | 000100 |
| Func | 100001 | 100011 | N/A | | | | |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| MemToReg | 0 | 0 | 0 | 0 | 1 | X | X |
| nPC\_Sel | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ExtOp | X | X | 0 | 0 | 1 | 1 | 1 |
| LuiOp | X | X | 0 | 1 | 0 | 0 | 0 |
| RegDst | 1 | 1 | 0 | 0 | 0 | X | X |
| ALUOp[1] | 0 | 0 | 1 | 0 | 0 | 0 | X |
| ALUOp[0] | 0 | 1 | 0 | 0 | 0 | 0 | X |

**（3）控制器逻辑表达式**

**\*附注：为简化表达式形式，以下采用Oi代表OpCode[i],Fi代表Func[i] 。**

**1.与门识别逻辑阵列**

**① **

**② **

**③ **

**④ **

**⑤ **

**⑥ **

**⑦ **

**2.或门生成逻辑阵列**

**① **

**② **

**③ **

**④ **

**⑤ **

**⑥ **

**⑦ **

**⑧ **

**⑨ **

**⑩ **

第五节 数据通路构造表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | NPC | PC | IM | GPR | | EXT | ALU | | DM | |
| WD | RD | A | B | ADDR | WD |
| addu | PC4 | NPC | PC | ALU | IM[`rd] |  | GPR[A] | GPR[B] |  |  |
| subu | PC4 | NPC | PC | ALU | IM[`rd] |  | GPR[A] | GPR[B] |  |  |
| ori | PC4 | NPC | PC | ALU | IM[`rt] | IM[`imm16] | GPR[A] | EXT |  |  |
| lui | PC4 | NPC | PC | ALU | IM[`rt] | IM[`imm16] | GPR[A] | EXT |  |  |
| beq | PC4 | NPC | PC |  |  | IM[`imm16] | GPR[A] | EXT |  |  |
| lw | PC4 | NPC | PC | DM | IM[`rt] | IM[`imm16] | GPR[A] | EXT | ALU |  |
| sw | PC4 | NPC | PC |  |  | IM[`imm16] | GPR[A] | EXT | ALU | GPR[B] |
| jal | imm[25:0] | NPC | PC | PC4 | 31 |  |  |  |  |  |
| jr | PC4 | GPR[rs] | PC |  |  |  | GPR[A] |  |  |  |
| 合并 | PC4| imm[25:0] | NPC| GPR[rs] | PC | ALU|DM| PC4 | IM[`rt]| IM[`rd]| 31 | IM[`imm16] | GPR[A] | EXT| GPR[B] | ALU | GPR[B] |
|

**第二章 测试验证**

第一节 测试代码

1、测试代码1

ori $a0,$0,0x1234

lui $a1,0x1234

nop

addu $a2,$a1,$a0

subu $a3,$a2,$a1

nop

ori $t0,$0,0

ori $t1,$0,1

ori $t2,$0,1

ori $t3,$0,2

beq $t1,$t2,target

sw $a3,0($t0)

beq $t1,$t3,target

sw $a3,4($t0)

target:

sw $a0,8($t0)

sw $a1,12($t0)

nop

sw $a2,16($t0)

sw $a3,20($t0)

lw $s0,8($t0)

lw $s1,12($t0)

lw $s2,16($t0)

lw $s3,20($t0)

beq $t1,$t3,end

sw $s0,24($t0)

sw $s1,28($t0)

end:

2、测试代码2

nop

ori $0,$0,1010

ori $1,$0,0x3ecf

ori $2,$0,0x8ace

nop

lui $3,0x1011

addu $4,$3,$2

addu $4,$4,$1

nop

addu $1,$1,$1

nop

subu $5,$4,$2

subu $5,$5,$1

jal work

ori $6,$0,1

ori $7,$0,1

nop

beq $6,$7,end

work:

sw $1,0($0)

sw $2,4($0)

nop

sw $3,8($0)

nop

sw $4,12($0)

lw $5,4($0)

sw $5,16($0)

end:

3、测试代码3

ori $1,$0,10

ori $2,$0,4

ori $3,$0,0

nop

ori $4,$0,0

ori $5,$0,1

lui $6,0x29ea

for\_1\_begin:

beq $4,$1,for\_1\_end

subu $6,$6,$2

nop

sw $6,0($3)

addu $3,$3,$2

addu $4,$4,$5

jal for\_1\_begin

for\_1\_end:

lw $7,8($0)

nop

4、测试代码4

ori $1,$0,0x2100

nop

nop

ori $2,$0,0x8ace

jal work

nop

lui $5,0x1092

addu $7,$4,$5

sw $7,0($0)

ori $8,$0,1

ori $9,$0,2

nop

ori $10,$0,1

sw $8,4($0)

sw $9,8($0)

nop

sw $10,12($0)

jal work2

nop

beq $11,$12,end

sw $4,16($0)

beq $11,$13,end

sw $5,20($0)

work:

addu $3,$1,$2

subu $4,$3,$1

subu $4,$4,$1

jr $ra

work2:

lw $11,4($0)

lw $12,8($0)

nop

nop

nop

lw $13,12($0)

jr $ra

end:

第二节 测试期望

1、测试期望1

（1）测试目的

测试基本指令{ori,lui}

测试运算R类指令{addu,subu}

测试装载类指令{lw,sw}

（2）测试原理

利用基本指令对寄存器赋值，验证其赋值的正确性后，可利用基本指令取base,再将运算类指令运算结果存入相应的内存地址，将内存中的数据读出到相应寄存器，即可同时验证运算类指令和装载类指令的正确性。

（3）测试预期结果

@00003000: $ 4 <= 00001234

@00003004: $ 5 <= 12340000

@0000300c: $ 6 <= 12341234

@00003010: $ 7 <= 00001234

@00003018: $ 8 <= 00000000

@0000301c: $ 9 <= 00000001

@00003020: $10 <= 00000001

@00003024: $11 <= 00000002

@00003038: \*00000008 <= 00001234

@0000303c: \*0000000c <= 12340000

@00003044: \*00000010 <= 12341234

@00003048: \*00000014 <= 00001234

@0000304c: $16 <= 00001234

@00003050: $17 <= 12340000

@00003054: $18 <= 12341234

@00003058: $19 <= 00001234

@00003060: \*00000018 <= 00001234

@00003064: \*0000001c <= 12340000

2、测试期望2

（1）测试目的

测试跳转类指令{beq,jal}

测试空指令{nop}

（2）测试原理

第一处为不成立的跳转条件，则其以下的sw指令会被执行，第二处为成立的跳转条件，其以下的sw指令不会被执行，直接跳转到target执行其后的sw指令。最终查验内存数据比对即可验证跳转类指令的正确性。

（3）测试预期结果

@00003004: $ 0 <= 000003f2

@00003008: $ 1 <= 00003ecf

@0000300c: $ 2 <= 00008ace

@00003014: $ 3 <= 10110000

@00003018: $ 4 <= 10118ace

@0000301c: $ 4 <= 1011c99d

@00003024: $ 1 <= 00007d9e

@0000302c: $ 5 <= 10113ecf

@00003030: $ 5 <= 1010c131

@00003034: $31 <= 00003038

@00003048: \*00000000 <= 00007d9e

@0000304c: \*00000004 <= 00008ace

@00003054: \*00000008 <= 10110000

@0000305c: \*0000000c <= 1011c99d

@00003060: $ 5 <= 00008ace

@00003064: \*00000010 <= 00008ace

3、测试期望3

（1）测试目的

综合测试支持指令集中全部指令。

（2）测试原理

原理综合，同前两次测试的原理。

（3）测试预期结果

@00003000: $ 1 <= 0000000a

@00003004: $ 2 <= 00000004

@00003008: $ 3 <= 00000000

@00003010: $ 4 <= 00000000

@00003014: $ 5 <= 00000001

@00003018: $ 6 <= 29ea0000

@00003020: $ 6 <= 29e9fffc

@00003028: \*00000000 <= 29e9fffc

@0000302c: $ 3 <= 00000004

@00003030: $ 4 <= 00000001

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9fff8

@00003028: \*00000004 <= 29e9fff8

@0000302c: $ 3 <= 00000008

@00003030: $ 4 <= 00000002

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9fff4

@00003028: \*00000008 <= 29e9fff4

@0000302c: $ 3 <= 0000000c

@00003030: $ 4 <= 00000003

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9fff0

@00003028: \*0000000c <= 29e9fff0

@0000302c: $ 3 <= 00000010

@00003030: $ 4 <= 00000004

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffec

@00003028: \*00000010 <= 29e9ffec

@0000302c: $ 3 <= 00000014

@00003030: $ 4 <= 00000005

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffe8

@00003028: \*00000014 <= 29e9ffe8

@0000302c: $ 3 <= 00000018

@00003030: $ 4 <= 00000006

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffe4

@00003028: \*00000018 <= 29e9ffe4

@0000302c: $ 3 <= 0000001c

@00003030: $ 4 <= 00000007

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffe0

@00003028: \*0000001c <= 29e9ffe0

@0000302c: $ 3 <= 00000020

@00003030: $ 4 <= 00000008

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffdc

@00003028: \*00000020 <= 29e9ffdc

@0000302c: $ 3 <= 00000024

@00003030: $ 4 <= 00000009

@00003034: $31 <= 00003038

@00003020: $ 6 <= 29e9ffd8

@00003028: \*00000024 <= 29e9ffd8

@0000302c: $ 3 <= 00000028

@00003030: $ 4 <= 0000000a

@00003034: $31 <= 00003038

@00003038: $ 7 <= 29e9fff4

4、测试期望4

（1）测试目的

综合测试支持指令集中全部指令。

（2）测试原理

原理综合，同前两次测试的原理。

（3）测试预期结果

@00003000: $ 1 <= 00002100

@0000300c: $ 2 <= 00008ace

@00003010: $31 <= 00003014

@0000305c: $ 3 <= 0000abce

@00003060: $ 4 <= 00008ace

@00003064: $ 4 <= 000069ce

@00003018: $ 5 <= 10920000

@0000301c: $ 7 <= 109269ce

@00003020: \*00000000 <= 109269ce

@00003024: $ 8 <= 00000001

@00003028: $ 9 <= 00000002

@00003030: $10 <= 00000001

@00003034: \*00000004 <= 00000001

@00003038: \*00000008 <= 00000002

@00003040: \*0000000c <= 00000001

@00003044: $31 <= 00003048

@0000306c: $11 <= 00000001

@00003070: $12 <= 00000002

@00003080: $13 <= 00000001

@00003050: \*00000010 <= 000069ce

**第三章 课后思考**

1、第一题

**根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？**

答：addr信号来自ALU模块的ALUOut端口。

因为是按字存储，地址都是4的倍数，地址最低两位缺省为0。

2、第二题

**在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？**

答：reset 对 PC、DM、GRF部件进行清零复位操作。

清零复位是为了模拟微处理器每次开机重启是的状态，指令从首地址开始，内存和寄存器堆清零。

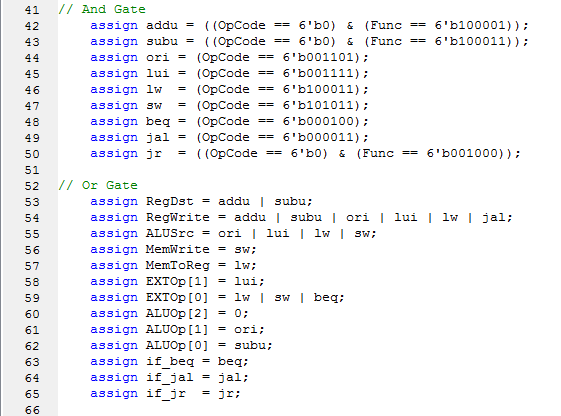
3、第三题

**列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例.**

答：

① （推荐）采用系统级描述方式。

例



② 采用门级描述方式。

例略。

③ 采用case语句编码方式。

例略。

4、第四题

**根据你所列举的编码方式，说明他们的优缺点。**

答：

① 表述简洁，清晰易懂。易于编码时对照指令集进行查验。可扩展性好。

② 优点：和与或门阵列一一对照，逻辑关系突出。但是代码冗余繁杂，不易编写。

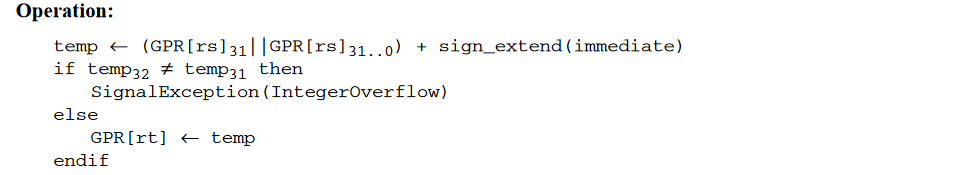
③ 对每一条输入的指令进行判断，从而决定了控制器的输出。但是不够直观。

5、第五题

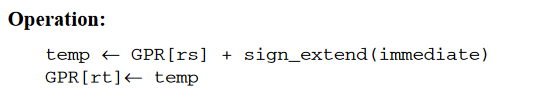
**C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。**

答：以addi 和addiu为例

Addi:



Addiu:



addi是将GPR[rs]的当前第31位与相加结果的第31位比较，如果相等证明没有溢出，那在没有溢出的前提下，GPR[rs]的当前第31位与相加结果的第31位一定是相等的，所以就是GPR[rt] ← temp,与addi同理。

add和addu同理。

6、第六题

**根据自己的设计说明单周期处理器的优缺点。**

答：优点：

逻辑简单，易于理解

缺点：

(1)单周期需要足够长的周期来完成最慢的指令（lw），即使大部分指令的速度都非常快。

(2)它需要3个加法器，一个用于ALU，两个用于PC的逻辑，而加法器是相对占用芯片面积的电路，尤其是如果他们的速度比较快。

(3) 它采用独立的指令存储器和数据存储器，而这在实际系统中是不现实的。大多数计算机有一个单独的大容量存储器来存储指令和数据，并且支持读和写的操作。

7、第七题

**简要说明jal、jr和堆栈的关系。**

答：在跳转到指定地址实现子程序调用的同时，需要将返回地址保存到$ra寄存器，即通常所说的“函数调用的现场保护”，以便子程序返回时能够继续调用之前的流程。对于跳转/分支指令，MIPS-CPU将自动保存$ra；若子程序需要嵌套调用其他子程序，则必须先存储$ra，通常是压入栈，子程序末尾弹出之前保存的$ra，然后jr实现跳转到$ra所指示PC地址.