北京航空航天大學

BEIHANG UNIVERSITY

计算机组成原理 Project3 实验报告

Logisim – 支持 8 指令单周期 CPU

北京航空航天大学

计算机学院

陈麒先

16061160

二〇一七年十一月

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！

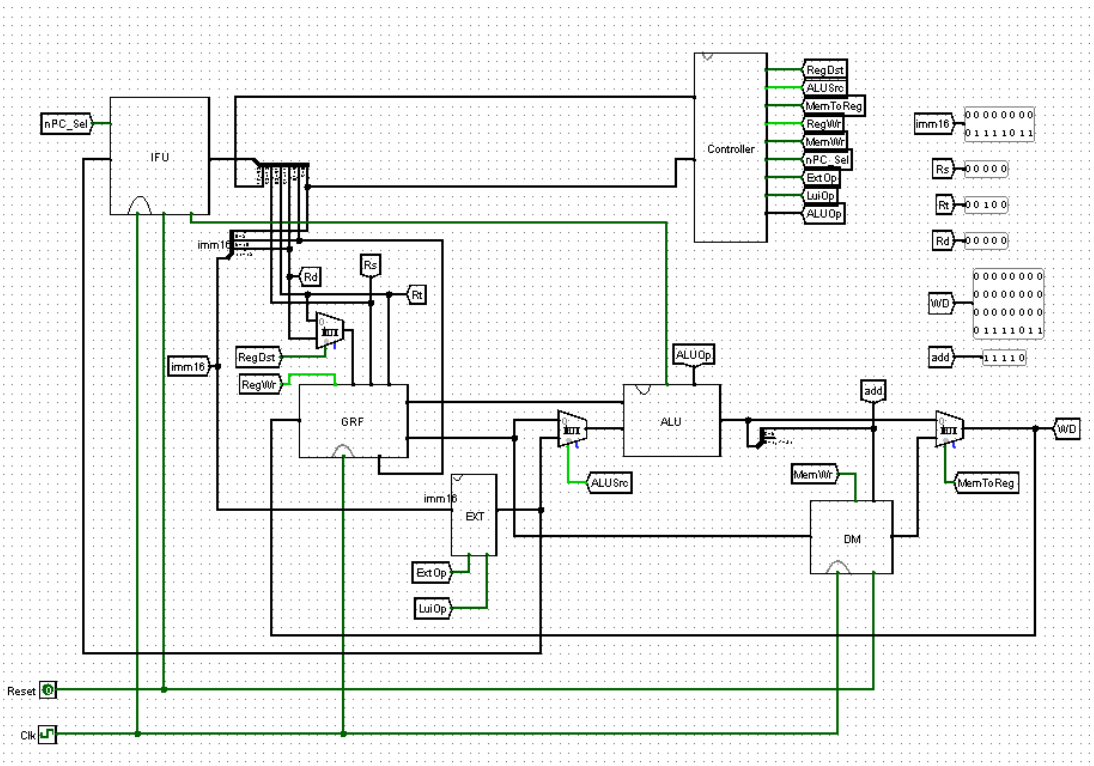
16061160

陈麒先

目录

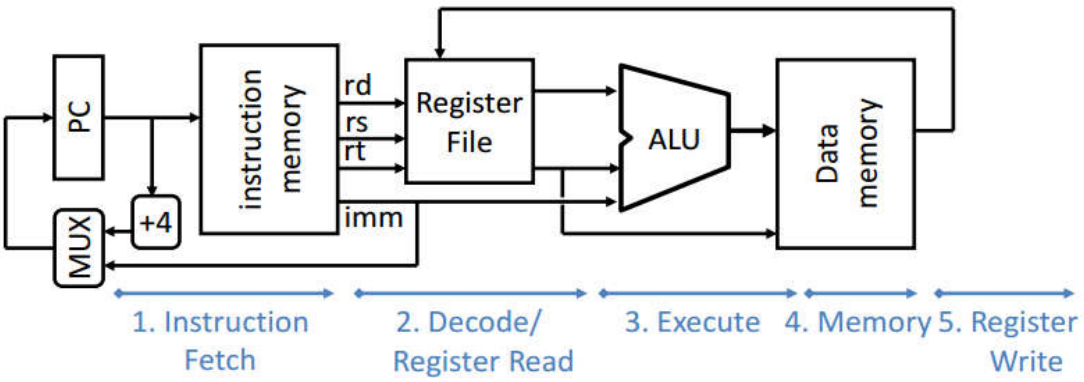
第一章 设计架构

第一节 单周期 CPU 顶层架构视图

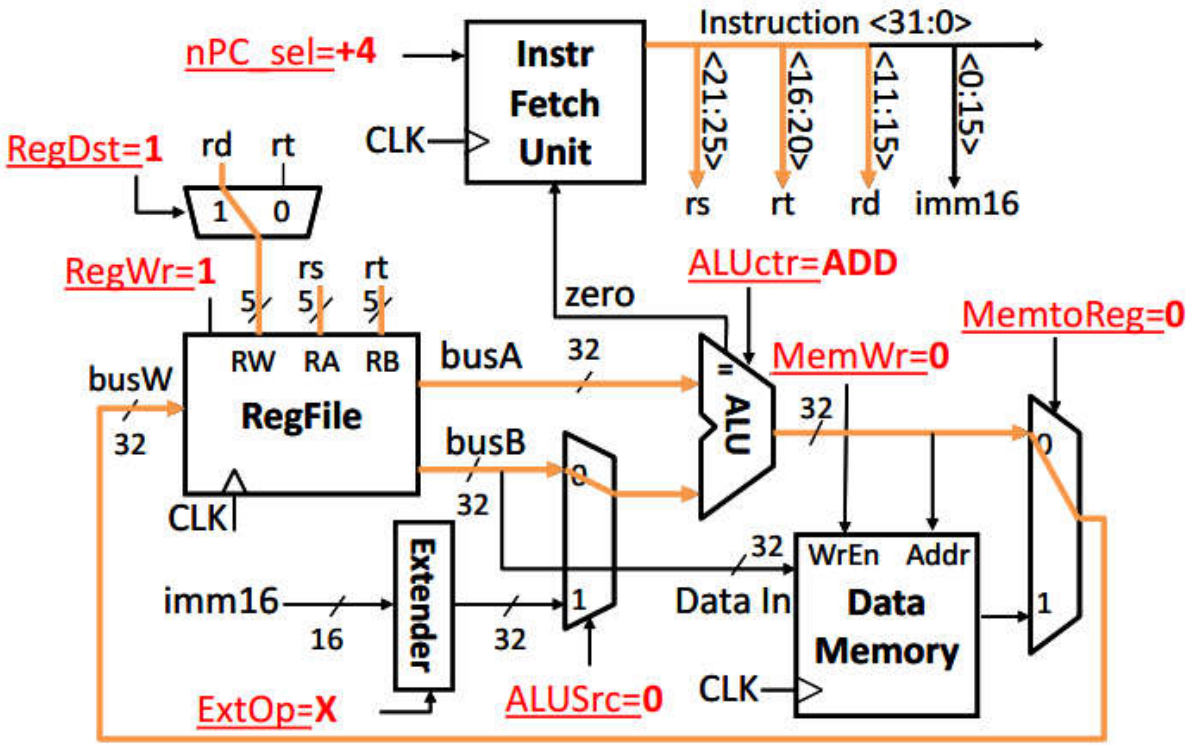


第二节 单周期 CPU 指令数据通路

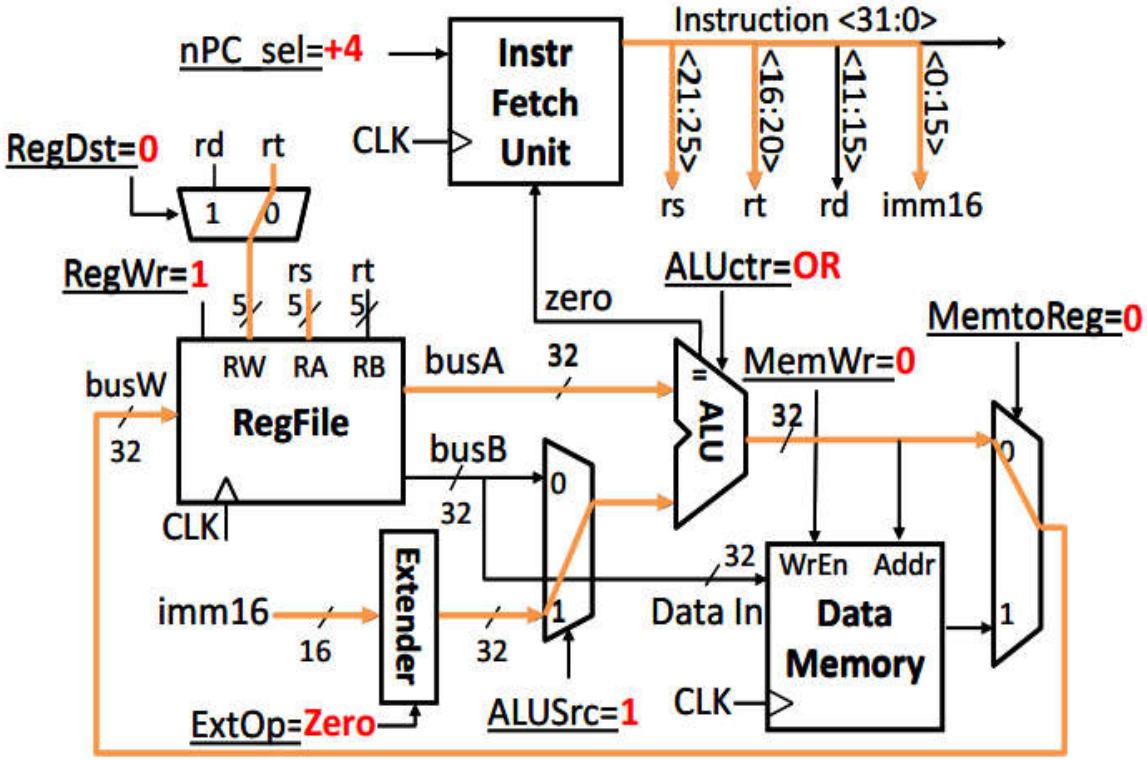
1、 数据通路整体架构



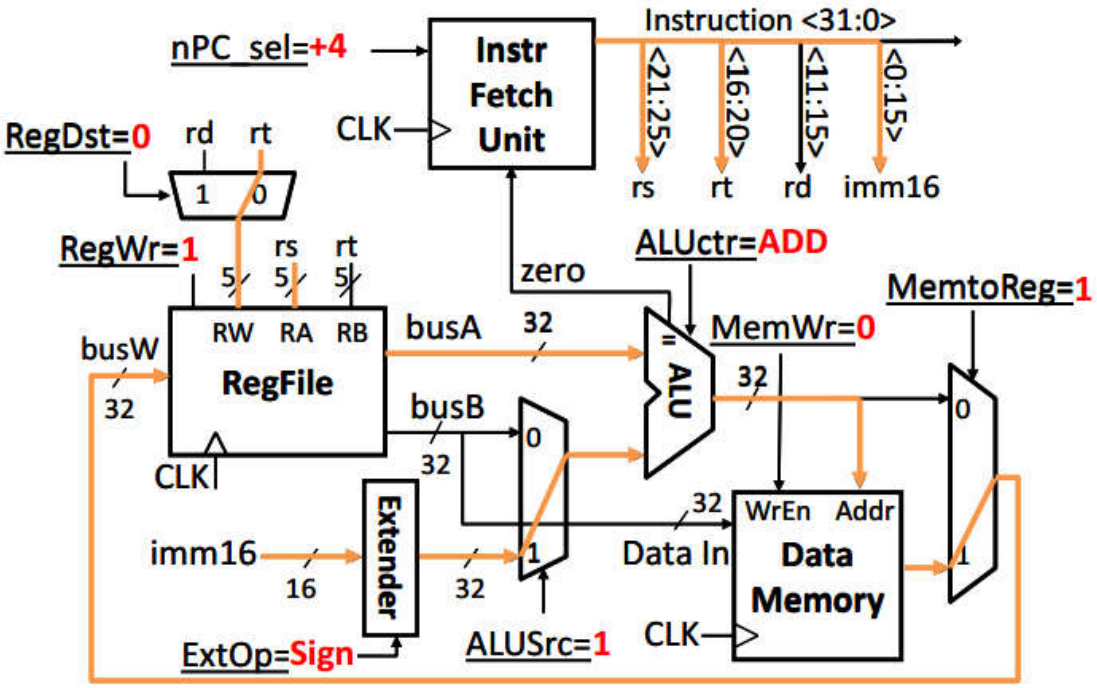
2、 addu / subu 指令数据通路



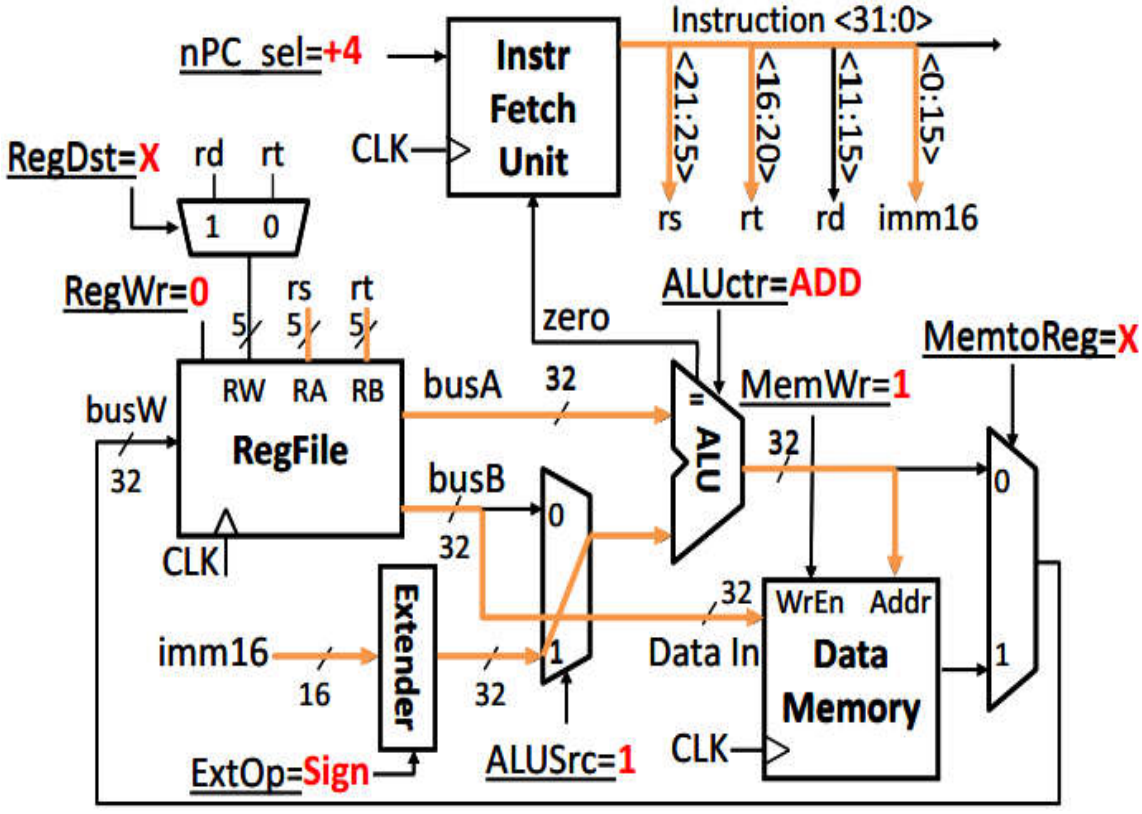
3、 ori 指令数据通路



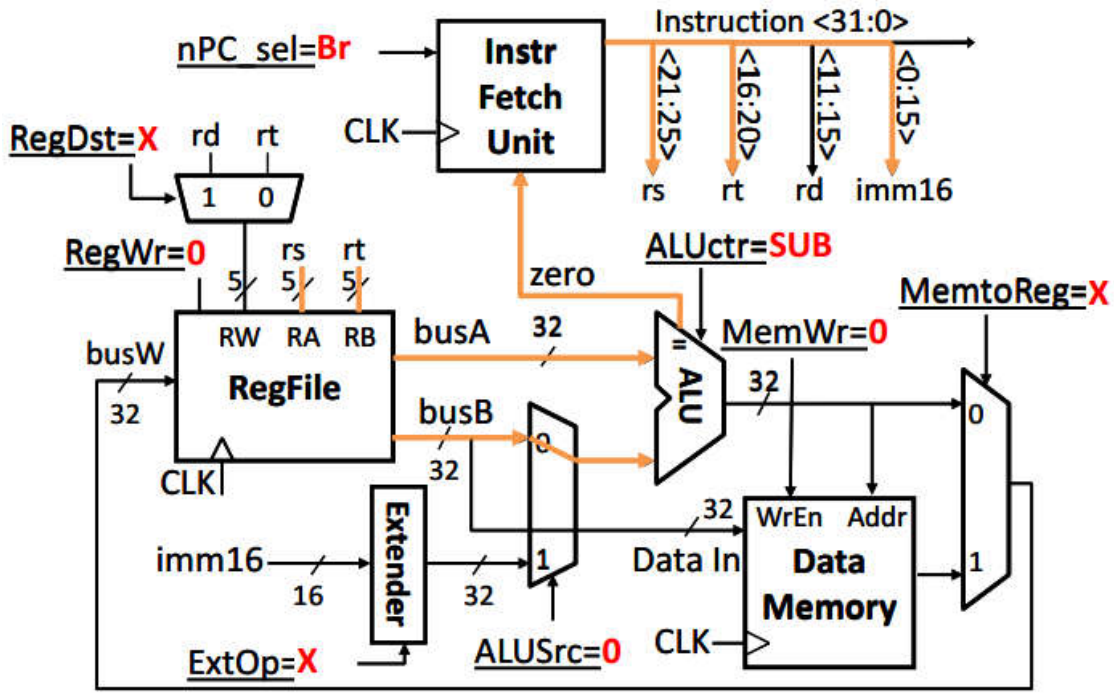
4、 lw 指令数据通路



5、 sw 指令数据通路



6、 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
nPC_Sel	In	跳转使能信号
zero	In	相等条件信号
Offset	In	地址偏移量
Ins	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、GRF

（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	异步复位	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
nPC_Sel	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 0i 代表 OpCode[i],Fi 代表 Func[i] 。

1.与门识别逻辑阵列

① $addu = \overline{O_5} \bullet \overline{O_4} \bullet \overline{O_3} \bullet \overline{O_2} \bullet \overline{O_1} \bullet \overline{O_0} \bullet F_5 \bullet \overline{F_4} \bullet \overline{F_3} \bullet \overline{F_2} \bullet \overline{F_1} \bullet F_0$

② $subu = \overline{O_5} \bullet \overline{O_4} \bullet \overline{O_3} \bullet \overline{O_2} \bullet \overline{O_1} \bullet \overline{O_0} \bullet F_5 \bullet \overline{F_4} \bullet \overline{F_3} \bullet \overline{F_2} \bullet F_1 \bullet F_0$

③ $ori = \overline{O_5} \bullet \overline{O_4} \bullet O_3 \bullet O_2 \bullet \overline{O_1} \bullet O_0$

④ $lui = \overline{O_5} \bullet \overline{O_4} \bullet O_3 \bullet O_2 \bullet O_1 \bullet O_0$

⑤ $lw = O_5 \bullet \overline{O_4} \bullet \overline{O_3} \bullet \overline{O_2} \bullet O_1 \bullet O_0$

⑥ $sw = O_5 \bullet \overline{O_4} \bullet O_3 \bullet \overline{O_2} \bullet O_1 \bullet O_0$

⑦ $beq = \overline{O_5} \bullet \overline{O_4} \bullet \overline{O_3} \bullet O_2 \bullet \overline{O_1} \bullet \overline{O_0}$

2.或门生成逻辑阵列

① $ALUSrc = ori + lui + lw + sw$

② $MemWrite = sw$

③ $R'egWrite = addu + subu + ori + lui + lw$

④ $MemToR'eg = lw$

⑤ $nPC\_Sel = beq$

⑥ $ExtOp = lw + sw + beq$

⑦ $LuiOp = lui$

⑧ $R'egDst = addu + subu$

⑨ $ALUOp[1] = ori$

⑩ $ALUOp[0] = subu$

第二章 测试验证

第一节 测试代码

1、测试代码 1

0x3404007b	1: ori $a0, $0, 123
0x348501c8	2: ori $a1, $a0, 456
0x3c06007b	3: lui $a2, 123
0x3c07ffff	4: lui $a3, 0xffff
0x34e7ffff	5: ori $a3, $a3, 0xffff
0x00868021	6: addu $s0, $a0, $a2
0x00c48823	7: subu $s1, $a2, $a0
0x34080000	8: ori $t0, $0, 0x0000
0xad040000	9: sw $a0, 0 ($t0)
0xad050004	10: sw $a1, 4 ($t0)
0xad060008	11: sw $a2, 8 ($t0)
0xad07000c	12: sw $a3, 12 ($t0)
0xad100010	13: sw $s0, 16 ($t0)
0xad110014	14: sw $s1, 20 ($t0)
0xad120018	15: sw $s2, 24 ($t0)
0x8d100010	16: lw $s0, 16 ($t0)
0x8d110014	17: lw $s1, 20 ($t0)
0x8d120018	18: lw $s2, 24 ($t0)

```
1  ori $a0, $0, 123
2  ori $a1, $a0, 456
3  lui $a2, 123
4  lui $a3, 0xffff
5  ori $a3, $a3, 0xffff
6  addu $s0, $a0, $a2
7  subu $s1, $a2, $a0
8  ori $t0, $0, 0x0000
9  sw $a0, 0 ($t0)
10 sw $a1, 4 ($t0)
11 sw $a2, 8 ($t0)
12 sw $a3, 12 ($t0)
13 sw $s0, 16 ($t0)
14 sw $s1, 20 ($t0)
15 sw $s2, 24 ($t0)
16 lw $s0, 16 ($t0)
17 lw $s1, 20 ($t0)
18 lw $s2, 24 ($t0)
```

2、测试代码 2

0x34080000	1: ori $t0, $0, 0
0x00000000	2: nop
0x34040001	3: ori $a0, $0, 1
0x00000000	4: nop
0x34050002	5: ori $a1, $0, 2
0x00000000	6: nop
0x34060000	7: ori $a2, $0, 0
0x00000000	8: nop
0x10800008	9: beq $a0, $0, target
0x00000000	10: nop
0xad040000	11: sw $a0, 0 ($t0)
0xad050004	12: sw $a1, 4 ($t0)
0x10c00004	13: beq $a2, $0, target
0x00000000	14: nop
0xad040008	15: sw $a0, 8 ($t0)
0xad05000c	16: sw $a1, 12 ($t0)
0x00000000	17: nop
0xad040010	18: target: sw $a0, 16 ($t0)
0xad050014	19: sw $a1, 20 ($t0)

```
1  ori $t0, $0, 0
2  nop
3  ori $a0, $0, 1
4  nop
5  ori $a1, $0, 2
6  nop
7  ori $a2, $0, 0
8  nop
9  beq $a0, $0, target
10 nop
11 sw $a0, 0 ($t0)
12 sw $a1, 4 ($t0)
13 beq $a2, $0, target
14 nop
15 sw $a0, 8 ($t0)
16 sw $a1, 12 ($t0)
17 nop
18 target: sw $a0, 16 ($t0)
19         sw $a1, 20 ($t0)
```

3、测试代码 3

0x34041234	1: ori $a0, $0, 0x1234
0x3c051234	2: lui $a1, 0x1234
0x00a43021	3: addu $a2, $a1, $a0
0x00c53823	4: subu $a3, $a2, $a1
0x34080000	5: ori $t0, $0, 0
0x34090001	6: ori $t1, $0, 1
0x340a0001	7: ori $t2, $0, 1
0x340b0002	8: ori $t3, $0, 2
0x112a0003	9: beq $t1, $t2, target
0xad070000	10: sw $a3, 0 ($t0)
0x112b0001	11: beq $t1, $t3, target
0xad070004	12: sw $a3, 4 ($t0)
0xad040008	14: sw $a0, 8 ($t0)
0xad05000c	15: sw $a1, 12 ($t0)
0xad060010	16: sw $a2, 16 ($t0)
0xad070014	17: sw $a3, 20 ($t0)
0x8d100008	18: lw $s0, 8 ($t0)
0x8d11000c	19: lw $s1, 12 ($t0)
0x8d120010	20: lw $s2, 16 ($t0)
0x8d130014	21: lw $s3, 20 ($t0)
0x112b0002	22: beq $t1, $t3, end
0xad100018	23: sw $s0, 24 ($t0)
0xad11001c	24: sw $s1, 28 ($t0)

```
1  ori $a0, $0, 0x1234
2  lui $a1, 0x1234
3  addu $a2, $a1, $a0
4  subu $a3, $a2, $a1
5  ori $t0, $0, 0
6  ori $t1, $0, 1
7  ori $t2, $0, 1
8  ori $t3, $0, 2
9  beq $t1, $t2, target
10 sw $a3, 0 ($t0)
11 beq $t1, $t3, target
12 sw $a3, 4 ($t0)
13 target:
14 sw $a0, 8 ($t0)
15 sw $a1, 12 ($t0)
16 sw $a2, 16 ($t0)
17 sw $a3, 20 ($t0)
18 lw $s0, 8 ($t0)
19 lw $s1, 12 ($t0)
20 lw $s2, 16 ($t0)
21 lw $s3, 20 ($t0)
22 beq $t1, $t3, end
23 sw $s0, 24 ($t0)
24 sw $s1, 28 ($t0)
25 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

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x00000000	0x0000007b	0x000001fb	0x007b0000	0xffffffff	0x007b007b	0x007aff85	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

寄存器结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x0000007b
$a1	5	0x000001fb
$a2	6	0x007b0000
$a3	7	0xffffffff
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x007b007b
$s1	17	0x007aff85
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000

2、测试期望 2

（1）测试目的

测试跳转类指令{beq}

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

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x00000000	0x00000001	0x00000002	0x00000000	0x00000000	0x00000001	0x00000002
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

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

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00000000	0x00000000	0x00000000	0x00001234	0x12340000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x12341234	0x00001234	0x00001234	0x12340000
0x00000000	0x00000000	0x00000000	0x00000000

寄存器结果：

$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00001234
$a1	5	0x12340000
$a2	6	0x12341234
$a3	7	0x00001234
$t0	8	0x00000000
$t1	9	0x00000001
$t2	10	0x00000001
$t3	11	0x00000002
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00001234
$s1	17	0x12340000
$s2	18	0x12341234
$s3	19	0x00001234
$s4	20	0x00000000

第三章 课后思考

1、第一题

若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。

答：30 位 PC 相比于 32 位 PC 而言：

优点：可以直接作为 IM 的地址输入进行取指。

缺点：每次计数器的值自增 1，PC ← PC + 4 的意义不明显。

破坏了设计的整体性，需要多引入 30 位寄存器这种元件，增加成本。

2、第二题

现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

答：

DM,和 GPR 的选择比较理想，IM 的选择可以改进。

对于 DM（数据内存）来说，选择的元件需要支持写入和读取操作。而 RAM（随机访问存储器）恰好课提供相应的功能。GRF 即为寄存器堆文件，故需使用寄存器，以保证对寄存器的读写正确性。

而 IM 从功能上讲，也需要对其进行读写相关操作。ROM（只读存储器）则无法对其进行写入。在我们的设计中需要预置好指令存储，在实际应用中，需要我们向处理器随时导入新的指令存储，因此，可以想办法将 IM 改进为同样支持写数据的元件，比如 RAM。

3、第三题

结合上文给出的样例真值表，给出 RegDst， ALUSrc， MemtoReg，RegWrite，nPC_Sel，ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

答：详见实验报告第一章第四节第（3）条 1、2 项，报告书第 9 页。

4、第四题

充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式，请给出化简后的形式。

答： 详见实验报告第一章第四节第（3）条 1、2 项，报告书第 9 页。

* 关于真值表中 X 的处理策略：

为了简化表达式，可统一将真值表中的 X 视为 0 处理。

5、第五题

事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由

答：空指令不会触发任何控制信号，即真值表中相关真值全 0，因此在处理器控制核心中无须对其进行特殊处理。

6、第六题

前文提到,“可能需要手工修改指令码中的数据偏移”,但实际上只需再增加一个 DM 片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

答：片选信号一般是在划分地址空间时，由逻辑电路产生的。可编程接口芯片都有一个片选开关，只有当该输入端处于有效电平时，接口芯片才进入电路工作状态，实现数据的输入输出。可设计 DM 片选信号有效，当且仅当地址从 0x3000 起始。

7、第七题

除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证 (Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

答：形式验证的优点：

①由于形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量。

②形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了模拟验证的不足。

③形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

形式验证的不足：

无法做到像测试一样大面积覆盖全部情况。