计算机组成原理实验报告

一、CPU 设计概述

（一）总体设计概述

本 CPU 为 logisim 实现的 32 位单周期 MIPS - CPU，支持的指令集包含{addu、subu、ori、sw、lw、beq、lui、nop}。为了实现这些功能，CPU 主要包含了 IM、GRF、DM、ALU、EXT、IFU、Controller 模块。

（二）关键模块定义及内部逻辑实现

1：IM

（1）端口定义

端口	方向/位数	功能
A	input [31:0]	输入指令地址 A
RD	Output [31:0]	输出指令 RD

（2）功能：通过输入当前指令的地址 A，得到当前指令 RD。

（3）内部逻辑：IM 使用 32bits*32 的 ROM 得以实现，所以实际的指令地址只有 5 位，所以截取地址 A 的 A[4:0]，得到实际地址，将其连接至 ROM 的 Address 端口处，获得指令 RD。如图 1 所示。

图 1：IM

2：GRF

（1）端口定义

端口	方向/位数	功能
A1	Input [4:0]	输入寄存器 1 编码 A1
A2	Input [4:0]	输入寄存器 2 编码 A2
A3	Input [4:0]	输入寄存器 3 编码 A3
WD3	Input [31:0]	输入待写入数据 WD3
RESET	Input 1	输入复位信号 RESET
CLK	Input 1	输入时钟信号 CLK

WE	Input 1	输入写操作控制信号 WE
RD1	Output [31:0]	输出寄存器 1 存储的值 RD1
RD2	Output [31:0]	输出寄存器 2 存储的值 RD2

（2）功能：通过输入的 A1、A2 和输出的 RD1、RD2 实现寄存器文件的读操作，通过输入的 A3 和 WD3 及控制信号 WE 实现寄存器文件的写操作，通过输入 RESET 实现寄存器文件的复位。

（3）内部逻辑：GRF 使用 32 个具有写功能的寄存器得以实现。对于读操作，采用两个多路选择器，以 A1，A2 为选择信号，以 32 个寄存器为待选择信号，实现寄存器文件的读取。对于写操作，首先将待写入数据 WD3 接入 32 个寄存器中，再采用一个译码器，以 A3 为编码，输出 32 个译码信号，再将该 32 个信号分别与写操作控制信号 WE 进行与操作，得到最终的写控制信号，从而实现写操作。对于复位操作，将输入信号 RESET 连接至 31 个寄存器的清零端口（除 0 号寄存器）。对于特殊寄存器 0 号寄存器，由于其值恒为 0，将其清零端口连接至常量 1。如图 2 所示。

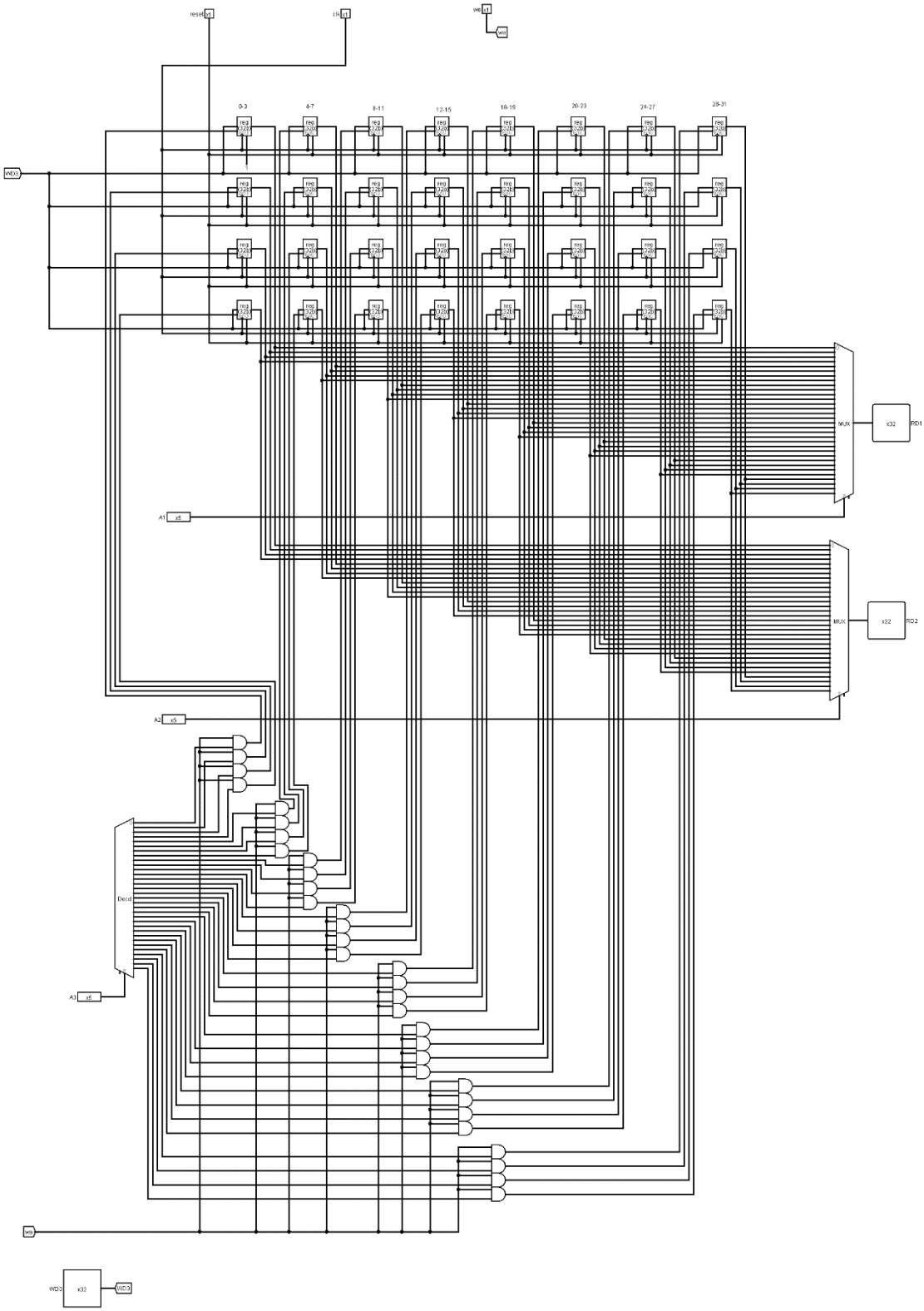

图 2：GRF

3：DM

（1）端口定义

端口	方向	功能
A	Input [4:0]	输入数据地址 A

WD	Input [31:0]	输入待写入数据 WD
CLK	Input 1	输入时钟信号 CLK
reset	Input 1	输入复位信号 reset
WE	Input 1	输入写操作控制信号 WE
RD	Output [31:0]	输出数据 RD

（2）功能：通过输入的 A 与输出的 RD 实现数据的读操作，通过输入的 A，WD 及写操作控制信号实现数据的写操作，通过输入的 reset 实现数据的复位操作。

（3）内部逻辑：DM 使用一个 32bits*32 的 RAM 得以实现。对于读操作，将输入的地址 A 和输出的数据 RD 分别连接至 RAM 的 Address 端口和 Data 端口，完成读操作。对于写操作，将输入的地址 A 和待写入的数据 WD 分别连接至 RAM 的 Address 端口和 Input 端口，再将写操作信号 WE 连接至 RAM 的 Store 端口，完成写操作。对于复位操作，将 reset 信号连接至 RAM 的 Clear 端口，完成复位操作。如图 3 所示。

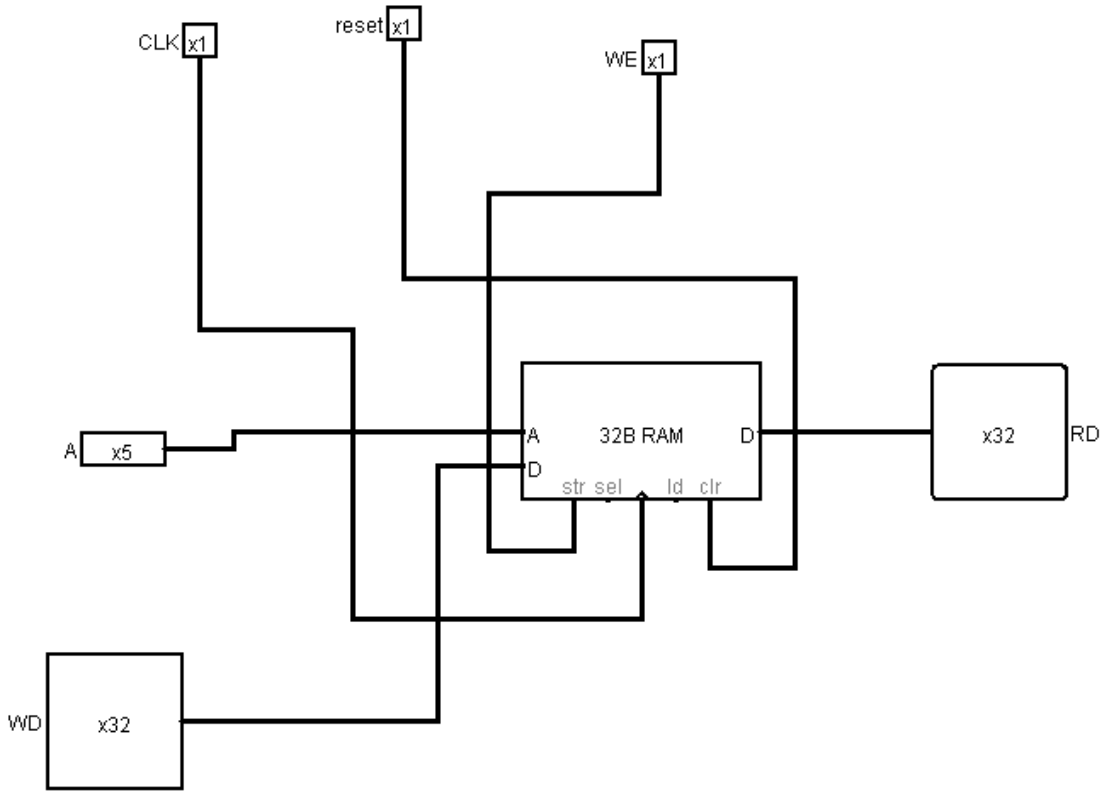

图 3：DM

4：ALU

（1）端口定义

端口	方向/位数	功能
A	Input [31:0]	输入操作数 A
B	Input [31:0]	输入操作数 B
OP	Input [2:0]	输入操作编码 OP
ANS	Output [31:0]	输出运算结果 ANS

（2）功能：通过输入的 A，B 和操作编码 OP，进行算数运算，输出结果 ANS。其中，OP=000 时，为加操作，OP=001 时，为减操作，OP=010 时，为或操作，OP=011 时，为大小比较。

（3）内部逻辑：ALU 使用多路选择器和算数运算器得以实现。通过加法器，减法器，或门和比较器，将 A，B 运算的结果作为多路选择器的输入，将操作编码 OP 作为选择信号，输出结果即为 ANS。如图 4 所示。

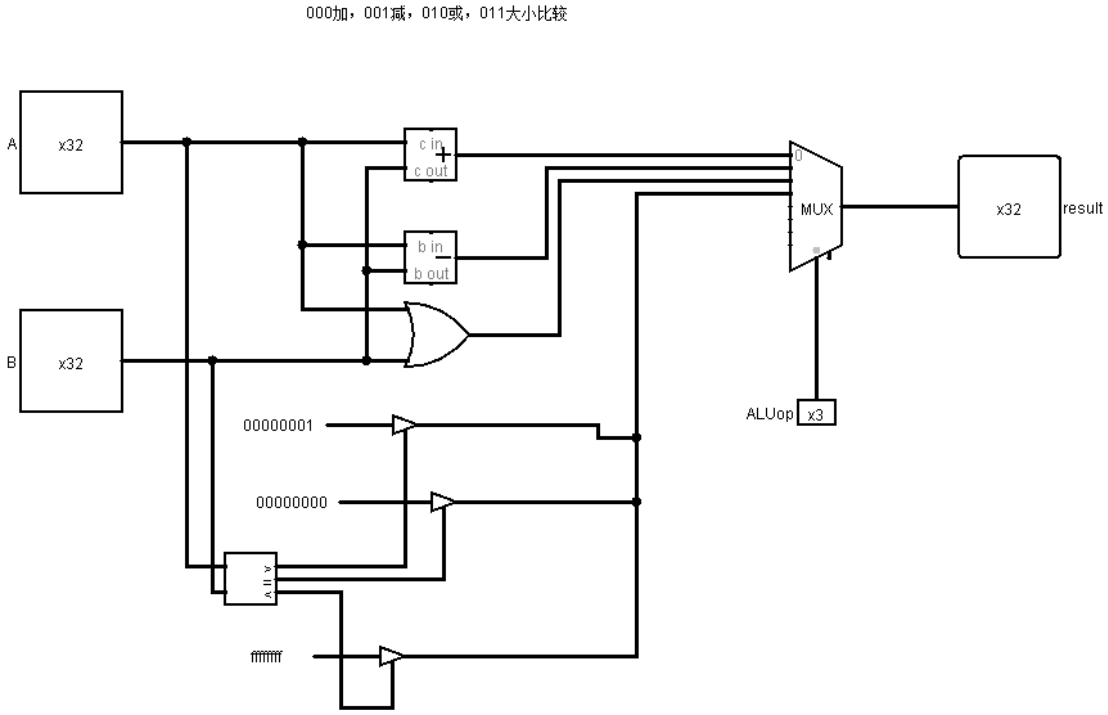

图 4：ALU

5：EXT

（1）端口定义

端口	方向/位数	功能
Imm	Input [15:0]	输入立即数 Imm
op	Input [1:0]	输入拓展编码 op
EXT(Imm)	Output [31:0]	输出拓展结果 EXT(Imm)

（2）功能：通过输入的立即数 Imm 和拓展编码 op，进行拓展，输出 EXT(Imm)为拓展结果。其中，op=00 时，为符号拓展，op=01 时，为零拓展，op=10 时，为加载高位。

（3）内部逻辑：EXT 使用多路选择器和 Bit Extender 得以实现。通过 Bit Extender 的符号拓展和零拓展以及 Splitter 器件，将 A，B 三种运算的结果作为多路选择器的输入，将拓展信号 op 作为选择信号，输出结果即为拓展结果 EXT(Imm)。如图 5 所示。

00符号拓展，01零拓展，10加载高位，11无所谓

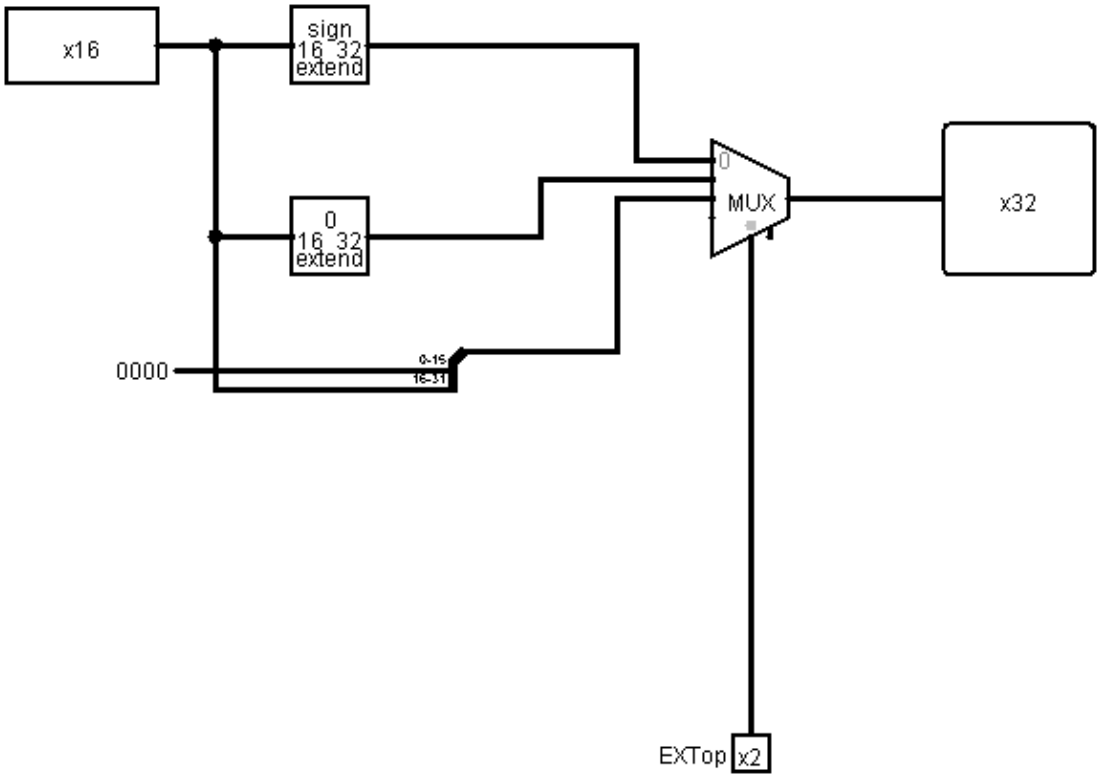

图 5：EXT

6：IFU

（1）端口定义

端口	方向/位数	功能
offset	Input [31:0]	输入的 PC 偏移量
CLK	Input 1	输入时钟信号 CLK
RESET	Input 1	输入复位信号 RESET
PCSrc	Input 1	输入 PC 选择信号 PCSrc
instr	Output [31:0]	输出指令 instr

（2）功能：通过 PC 寄存器中的指令地址，得到输出的指令 instr。通过输入的 PC 偏移量 offset 和 PC 选择信号 PCSrc,得到下一周期的 PC 值。

（3）内部逻辑：IFU 使用 IM 模块，加法器和多路选择器得以实现。对于指令获取操作，采用 IM 模块，将 PC 寄存器中的值连接至 IM 模块的 A 端口，得到 IM 模块的输出 RD，即为输出的指令 instr。对于 PC 的更新操作，通过加法器得到 PC 的两个更新值作为多路选择器的输入，将 PCSrc 作为多路选择器的选择信号，得到的结果即为下一周期的 PC 值。如图 6 所示。

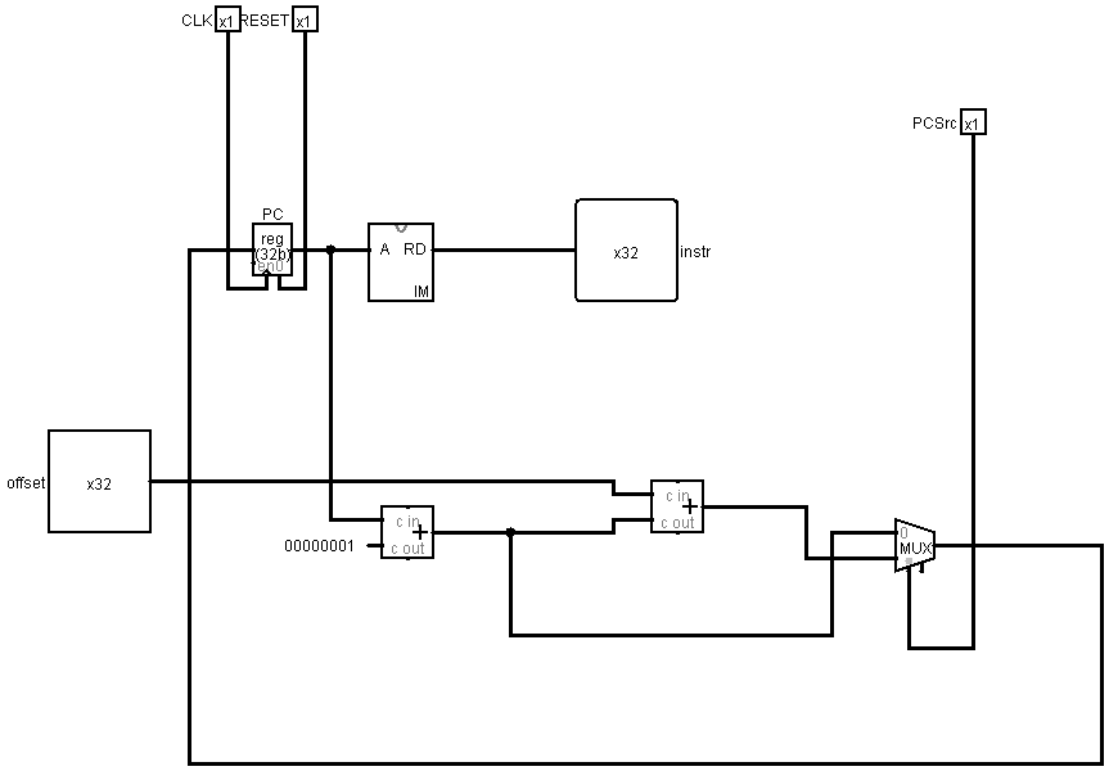

图 6：IFU

7：Controller：详见（三）

（三）CPU 控制中心——Controller

1：端口定义

端口	方向/位数	功能
op	Input [5:0]	输入的指令操作码 op
func	Input [5:0]	输入的指令函数 func
Branch	Output 1	输出的分支信号 Branch
EXTop	Output [1:0]	输出的拓展信号 EXTop
ALUop	Output [2:0]	输出的运算信号 ALUop
MemWrite	Output 1	输出的数据写信号 MemWrite
RegWrite	Output 1	输出的寄存器写信号 MemWrite
Reg&Mem	Output 1	输出的寄存器和内存选择信号 Reg&Mem
Data&offset	Output 1	输出的 offset 选择信号 Data&offset
RegDst	Output 1	输出的写寄存器编码选择信号 RegDst
ALUSrc	Output 1	输出的运算数选择信号 ALUSrc

2:**功能：**通过输入信号 op 和 func 的译码，得到指令的类型，再根据类型确定 CPU 内各个控制信号的输出。

3：指令与各控制信号的真值表

	ADDU	SUBU	ORI	LW	SW	BEQ	LUI

op	000000	000000	001101	100011	101011	000100	001111
func	100001	100011	xxxxxx	xxxxxx	xxxxxx	xxxxxx	xxxxxx
Branch	0	0	0	0	0	1	0
EXTop	xx	xx	01	00	00	xx	10
ALUop	000	001	010	000	000	011	xxx
MemWrite	0	0	0	0	1	0	0
RegWrite	1	1	1	1	0	0	1
Reg&Mem	0	0	0	1	x	x	x
Data&offset	0	0	0	0	X	x	1
RegDst	0	0	1	1	x	x	1
ALUSrc	0	0	1	1	1	x	x

4：**内部逻辑**：Controller 使用 Splitter，与门和或门得以实现。Controller 分为两步，第一步通过 op 各分量和 func 各分量的与操作确定具体指令，第二步通过对具体指令的或操作确定各控制信号。如图 7 所示。

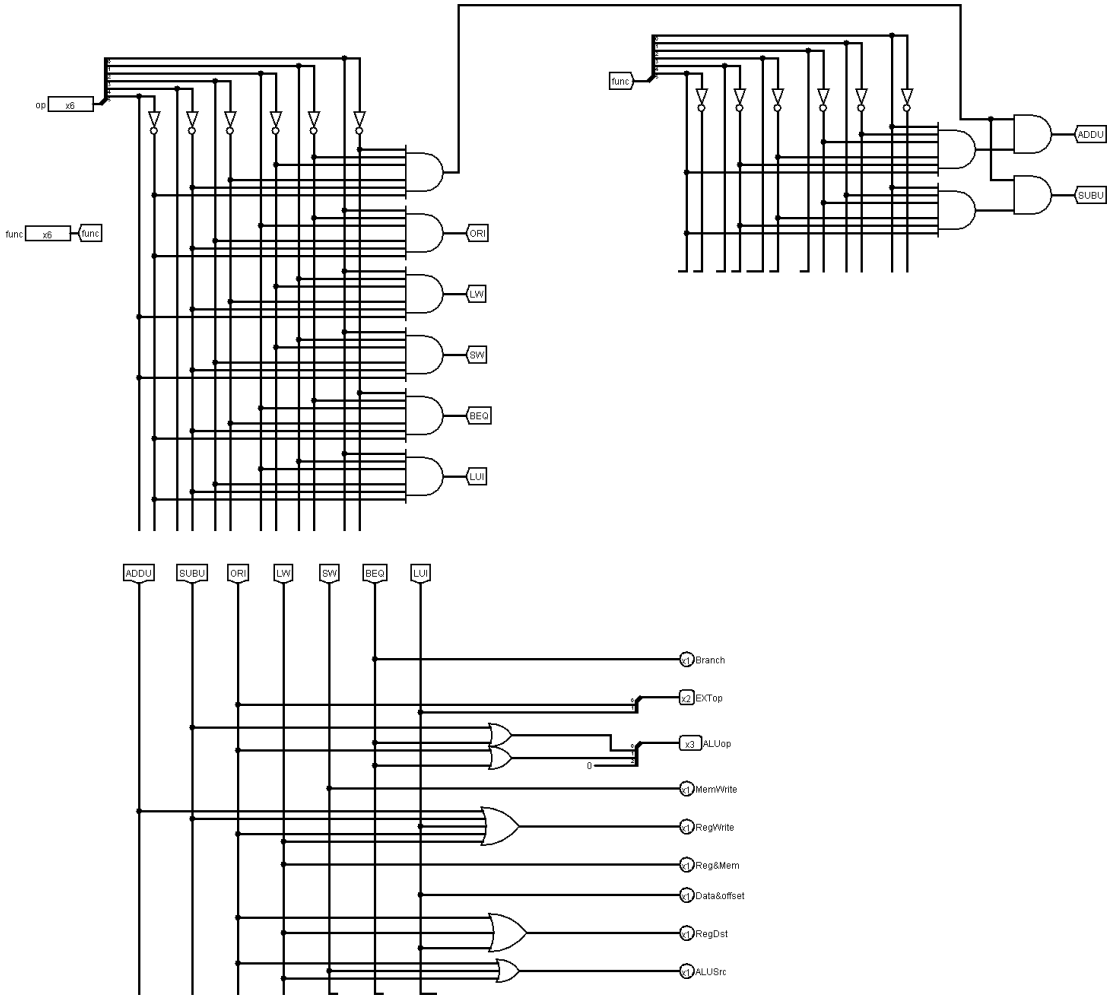

图 7：Controller

（四）CPU 组成

1：CPU 由各模块和各控制信号组成，如图 8 所示。

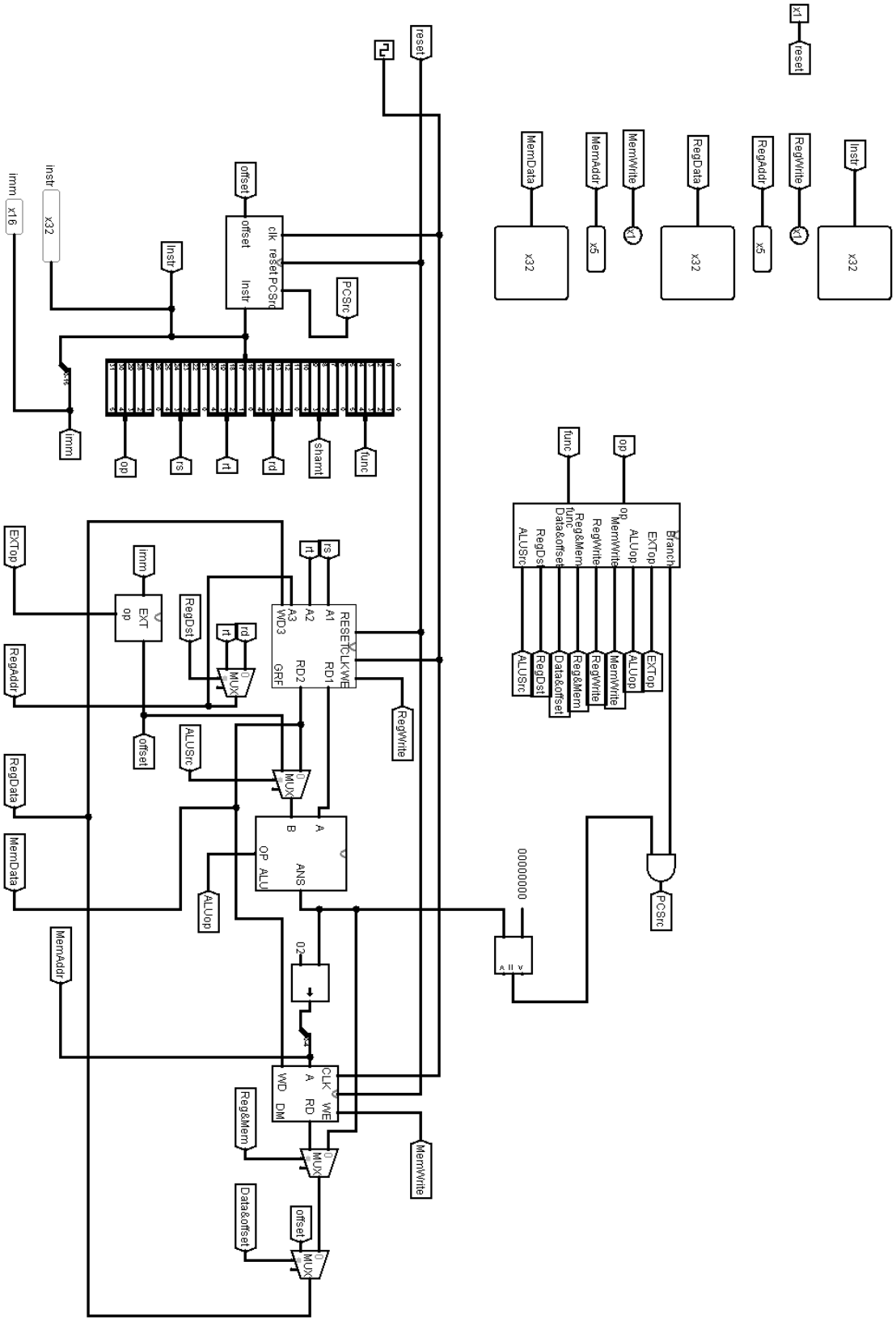

图 8：CPU

2：各模块作用

（1）IFU：读取指令；
（2）GRF：读写寄存器数据；

（3）EXT：拓展立即数；
（4）ALU：运算；
（5）DM：读写内存数据；
（6）Controller：输出控制信号。

3：各控制信号的功能

（1）Branch：若 Branch 为 1，表示指令为 beq，且若 ALU 运算得到的结果为 0，则控制信号 PCSrc 为 1，否则为 0；
（2）EXTop：该信号连接至 EXT 模块的 op 端口，控制 EXT 的拓展方式；
（3）ALUop：该信号连接至 ALU 模块的 OP 端口，控制 ALU 的运算方式；
（4）MemWrite：该信号连接至 DM 模块的 WE 端口，控制内存数据的写入；
（5）RegWrite：该信号连接至 GRF 模块的 WE 端口，控制寄存器数据的写入；
（6）Reg&Mem：该信号连接至一个多路选择器的选择端口，若为 0，选择 ALU 的运算结果，若为 1，选择从内存文件中读取的数据；
（7）Data&offset：该信号连接至一个多路选择器的选择端口，若为 0，选择上一个多路选择器的输出数据，若为 1，选择经拓展后的立即数；
（8）RegDst: 该信号连接至一个多路选择器的选择端口，若为 0，选择寄存器 rd 作为目的寄存器，若为 1，选择寄存器 rt 作为目的寄存器；
（9）ALUSrc：该信号连接至一个多路选择器的选择端口，若为 0，选择 GRF 模块输出的寄存器数据 RD2 作为 ALU 模块的操作数 B，若为 1，选择经拓展后的立即数作为 ALU 模块的操作数 B；
（10）PCSrc：该信号连接至 IFU 模块的 PCSrc 端口，控制下一周期 PC 寄存器的值。

二、测试方案

（一）策略：先对不依靠其他指令的基本指令测试，如 ori、lui，确保正确后再对其他单条指令进行测试，最后复杂测试包括所有指令。

（二）简单测试样例

1.ORI 指令

```
ori $a0,$0,123
ori $a1,$a0,456
```

2.LUI 指令

```
lui $a2,123
lui $a3,0xffff
```

3.ADDU 指令

```
ori $a0,$0,123
ori $a1,$a0,456
```

```
addu $s0,$a0,$a1
addu $s1,$a0,$s0
```

4.SUBU 指令

```
ori $a0,$0,123
ori $a1,$a0,456
subu $s0,$a1,$a0
subu $s1,$a0,$s0
```

5.SW 指令

```
ori $a0,$0,123
ori $a1,$a0,456
ori $t0,$0,0x0000
sw $a0,0($t0)
sw $a1,4($t0)
```

6.LW 指令

```
ori $a0,$0,123
ori $a1,$a0,456
ori $t0,$0,0x0000
sw $a0,0($t0)
sw $a1,4($t0)
lw $a2,4($t0)
lw $a3,0($t0)
```

7.BEQ 指令

```
ori $a0,$0,1
ori $a1,$0,2
ori $a2,$0,1
ori $t0,$0,0x0000
beq $a0,$a1,loop1
beq $a0,$a2,loop2
loop1:sw $a0,0($t0)
loop2:sw $a1,4($t1)
```

8.NOP 指令

```
ori $a0,$0,1
nop
ori $a1,$0,2
nop
ori $a2,$0,1
nop
```

（三）复杂测试样例

```
1：ori $s0,$0,10 #s0=n=10
    ori $s1,$0,1  #s1=1
    ori $t0,$0,0  #t0=i=0
    ori $t1,$0,0  #t1=sum=0
    ori $t2,$0,0  #t2=fusum=0
    for_1_begin:
        beq $t0,$s0,for_1_end
        nop
        addu $t1,$t1,$t0
        subu $t2,$t2,$t0

        addu $t0,$t0,$s1
        beq $t0,$t0,for_1_begin
        nop
    for_1_end:
        ori $s2,$0,0x0000
        sw $t1,0($s2)      #t1=45
        sw $t2,4($s2)      #t2=-45
        lw $s3,4($s2)      #s3=-45
        lw $s4,0($s2)      #s4=45
        lui $t3,0xffff
        ori $t3,$t3,0xffff #t3=-1
        addu $t4,$s3,$t3  #t4=-46
        addu $t5,$s4,$t3  #t5=44
2：ori $s0,$0,10 #s0=n=10
    ori $s1,$0,1  #s1=1
    ori $s2,$0,4  #s2=4
    lui $s3,0xffff
    ori $s3,$s3,0xffff#s3=-1
    ori $t1,$0,0x0000
    ori $t0,$0,0  #t0=i=0
    for_1_begin:
        beq $t0,$s0,for_1_end
        nop
        sw $t0,0($t1)
        addu $t1,$t1,$s2
        addu $t0,$t0,$s1
        beq $t0,$t0,for_1_begin
        nop
    for_1_end:
    subu $t0,$s0,$s1 #t0=i=n-1
    for_2_begin:
```

```
        beq $t0,$s3,for_2_end
        nop
        subu $t1,$t1,$s2
        lw $s4,0($t1)
        subu $t0,$t0,$s1
        beq $t0,$t0,for_2_begin
        nop
    for_2_end:
```

三、思考题

（一）现在我们的模块中 IM 使用 ROM，DM 使用 RAM， GRF 使用 Register，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：IM 使用 ROM 使得指令存储器只读，这是不合理的，因为在实际处理器中，我们必须将程序的机器码写入指令存储器，所以指令存储器也应该使用 RAM。DM 使用 RAM 合理，因为我们需要对数据进行读和写操作。GRF 使用 Register 是部分合理的，因为 GRF 中不止 32 个通用寄存器，还有其他寄存器，我们这里使用 32 个寄存器表示 GRF 是简略表示。

（二）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：nop 未加入真值表时，由于控制信号的输出逻辑，当当前指令为 nop 时，控制信号 MemWrite 与 RegWrite 为 0 ，寄存器文件 GRF 与数据内存 DM 并不会发生改变，相当于什么都没做，仅仅是延迟了一个周期，这就是指令 nop 的功能，所以不需要将 nop 加入到控制信号真值表中。

（三）上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

答：假设 DM 的起始地址为 0x0000_2000，那么只需要将数据地址的[15:12]位与 0x2 比较，比较结果就是片选信号。相当于假设有两块 RAM，一块从 0x0000_0000 开始，一块从 0x0000_2000 开始，每块 RAM 里面的内存就是从 0x0000 开始，而实际上只有第二块 RAM，数据地址也都大于等于

0x0000_2000，所以省去了手工修改的麻烦。为了解决这个问题，我将 MARS 的地址设为 Data at Address 0。这样 DM 的起始地址为 0x0000_0000,IM 的起始地址为 0x0000_3000。而对于现有的指令而言，指令地址 PC 只有两种的变化，加 4，加 4 加 offset。这两种方式得到的下一周期的 PC 值都是相对的，故起始地址为 0x0000_3000 就目前指令集而言，与起始地址为 0x0000_0000 相同。

（四）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，简要阐述相比于测试，形式验证的优劣之处。

答：优点：(1)形式验证是对指定描述的所有可能的情况进行验证，有效克服了测试验证的不足；

(2)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，不需要考虑如何获得测试向量；

(3)形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

缺点：形式验证到目前为止仍然不能有效地验证电路的性能，如电路的时延和功耗等。