# 2021年北航计组P3课下设计报告

## 一 CPU设计方案简述

### （一）总体设计概述

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

### （二）关键模块设计

#### 1. GRF

**GRF端口定义**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 复位信号， 1 复位，0无效 |
| WE | I | 写使能信号，1写入，0不能写入 |
| A1 | I | 5位地址输入信号，将对应的寄存器数据读出到RD1 |
| A2 | I | 5位地址输入信号，将对应的寄存器数据读出到RD2 |
| A3 | I | 5位地址输入信号，将数WD存储到所对应的寄存器 |
| WD | I | 32位输入数据 |
| RD1 | O | 输出A1地址所对应的寄存器数据 |
| RD2 | O | 输出A2地址所对应的寄存器数据 |

**功能定义**

| 序号 | 功能名称 | 描述 |
| --- | --- | --- |
| 1 | 复位 | reset信号有效时，所有寄存器存储的数据清零 |
| 2 | 读数据 | 读A1,A2所对应的寄存器数据到RD1，RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿到来时，将WD存入到A3所对应的寄存器 |

#### 2. DM

**DM端口定义**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| A [4：0] | I | 读入五位地址信号 |
| WD [31:0] | I | 32位信号，存入A所对应的地址 |
| clk | I | 时钟信号 |
| reset | I | 清零信号 |
| WE | I | 读入使能信号，当1时，存入数据 |
| RD [31:0] | O | 32位信号，输出A所对应的寄存器的数据 |

**DM功能描述**

| 序号 | 功能名称 | 描述 |
| --- | --- | --- |
| 1 | 清零 | 当reset信号位1时，RAM里的数据清零 |
| 2 | 读入数据 | 当WE信号为1，将WD存储到A对应的RAM地址 |

**备注：本模块实际接入的地址信号经过ALU，信号宽度为32位，在模块中，用splitter选取2-6位的信号作为5位输入信号。**

## 3.ALU

**ALU 端口简述**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| DataA [31:0] | I | 数据A |
| DataB [31:0] | I | 数据B |
| ALUctrl [2:0] | I | 运算控制选择信号 |
| result [31:0] | O | 输出结果 |
| ALUzero | O | 当result结果为0，ALUzero输出为1 |

**ALU 功能介绍**

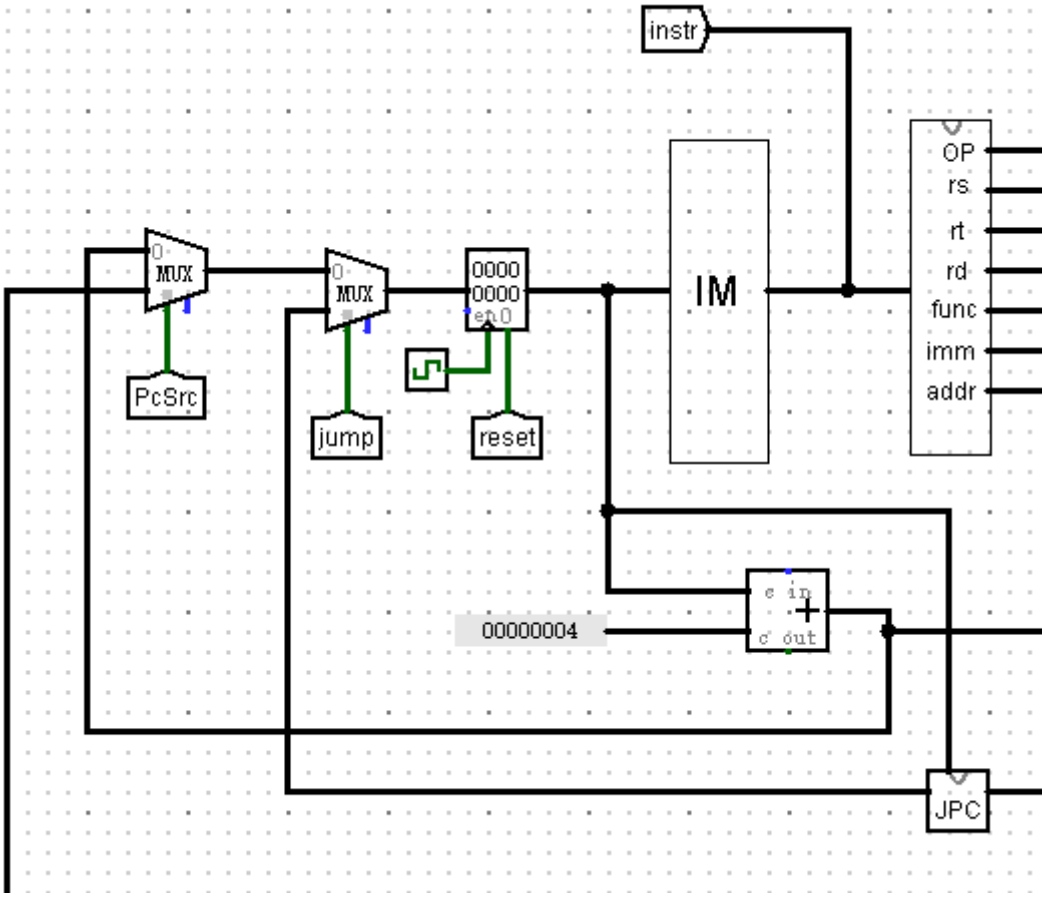
| ALUctrl | 运算描述 |
| --- | --- |
| 000 | A&B |
| 001 | A\|B |
| 010 | A + B |
| 011 | A - B |
| 100 | A⊕B |
| 101 | A=B? （unsigned） |

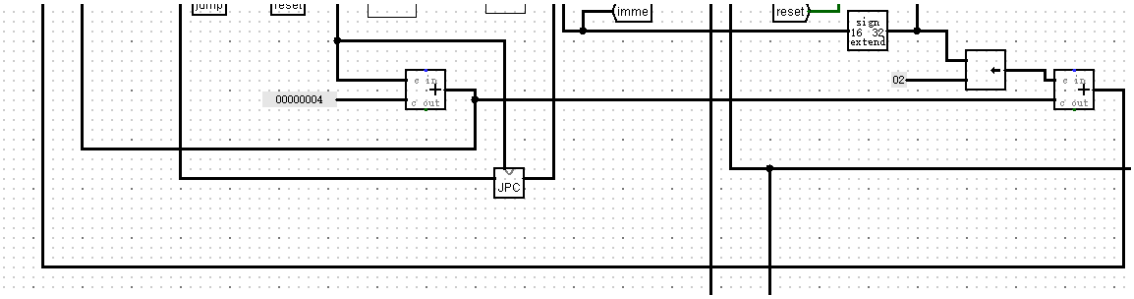
## 4. IFU

IFU模块主要由寄存器、IM子模块（存储指令）以及数据通路构成。

IFU功能定义：

1. 复位功能，当reset信号被置为1时，寄存器数据清零。
2. 取指令，根据当前的地址PC，从IM模块中取出指令。
3. 计算下一指令，一般下一条指令PC都为 PC+4，若当前指令为j指令，则下一指令地址为PC □ PC31..28|| （instr_index<<2)，若当前指令为beq指令，则下一指令为PC = PC + 4 + sign_extend(offset<<2)




备注：图中JPC模块计算j指令的转跳地址。

## 5.控制器

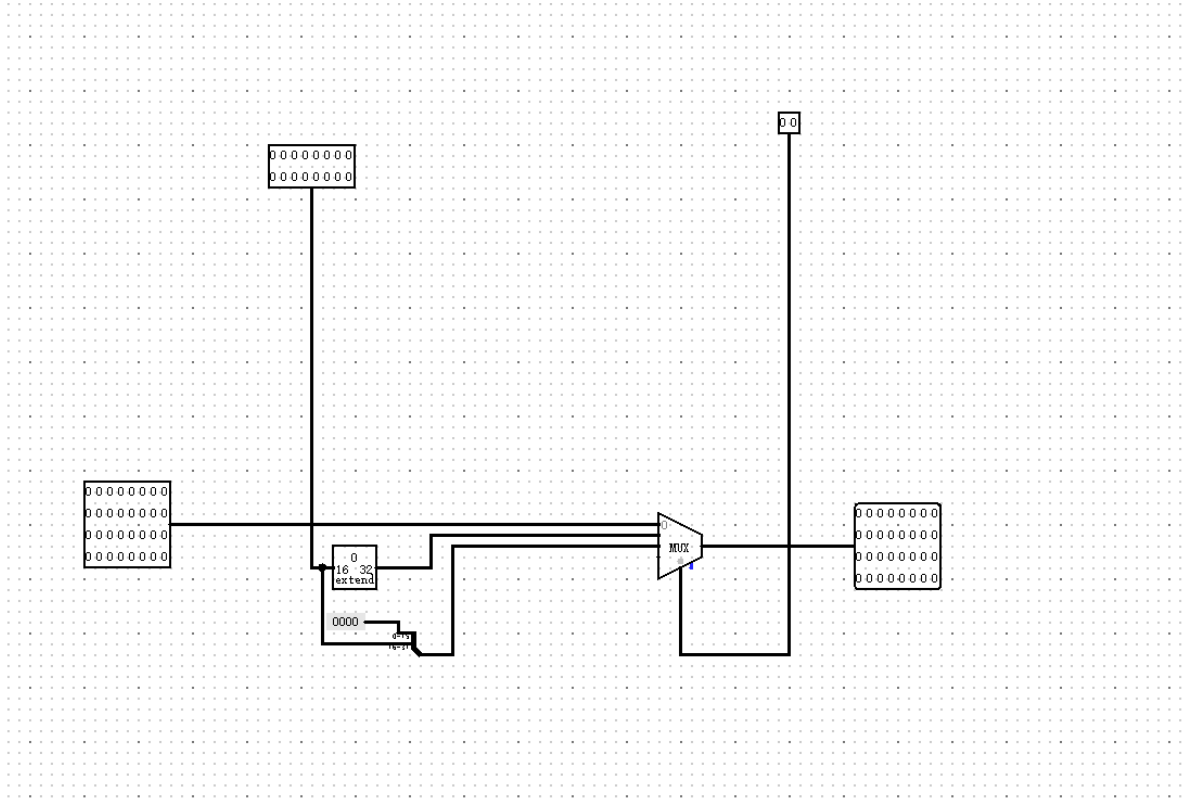
| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| opcode [5:0] | I | 6位opcode输入信号 |
| func [5:0] | I | 6位func输入信号 |
| MemWrite | O | 存储器写入使能信号 |
| RegWrite | O | 寄存器写入信号 |
| RegDst | O | 选择GRF A3端输入信号为Rt或Rd |
| ALUSrc | O | 选择ALU B端读入信号为RD2或立即数 |
| MemtoReg | O | 选择WD3读入信号直接由ALU得到还是寄存器的读数 |
| ALUctrl [2:0] | O | ALU的控制信号 |
| branch | O | beq指令的控制信号 |
| jump | O | j指令的控制信号 |
| Ext [1:0] | O | 实现ori和Lui指令扩充的控制信号 |

## 6. EXT

功能介绍

基于控制器所给Ext信号，选择对应的ALU的B端输入

| Ext输入 | 输出描述 |
| --- | --- |
| 00 | 输出除Ori、Lui以外的指令所需的B端输入 |
| 01 | 输出ori所需的0扩展后的32位立即数 |
| 10 | Lui所需的高位为16位立即数的32位输出 |




# (三) 重要机制实现方法

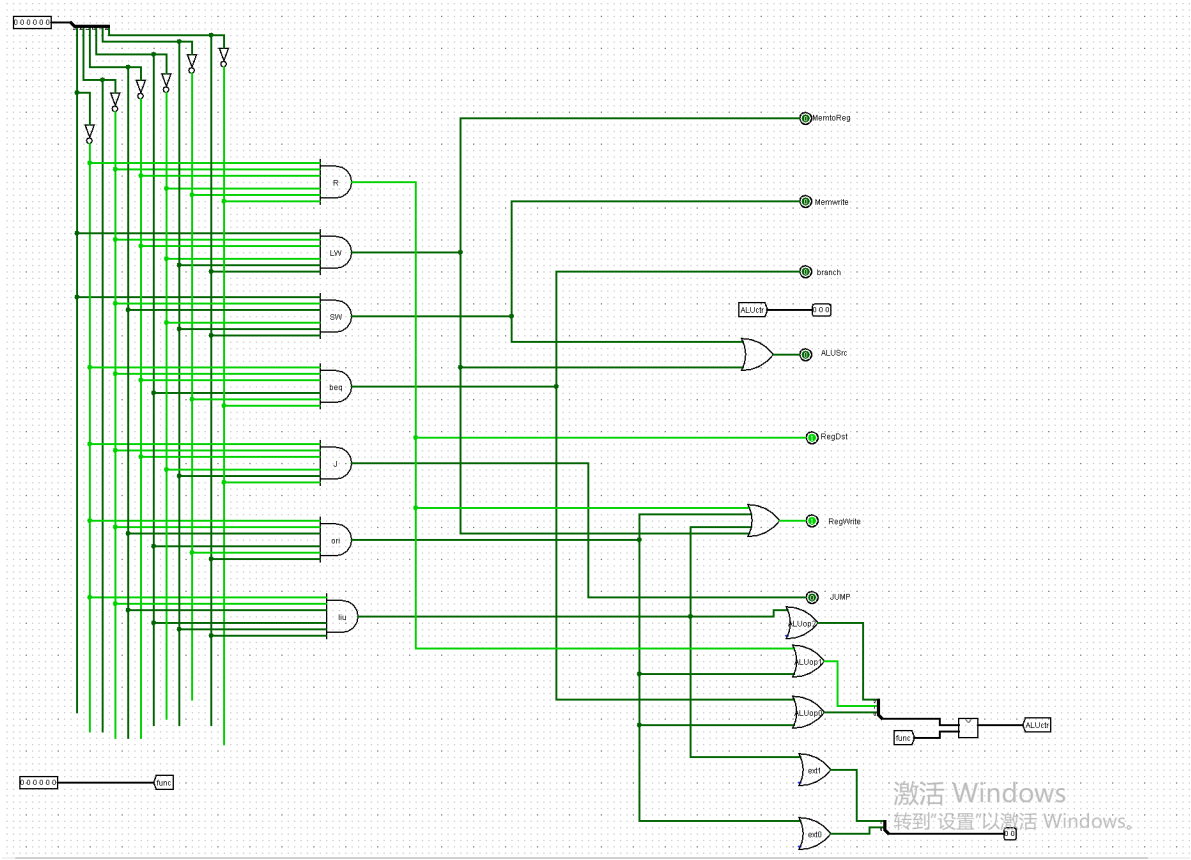
控制器主要由主译码器实现

## 主译码器设计

主译码器输出各种控制信号，ALUctrl信号不由主译码器模块直接得到，主译码器输出ALUop信号给ALU译码器子模块，ALUctr控制信号通过ALU译码器子模块由ALUop与func输入信号得到。

### 主译码器真值表

| 控制信号 | R | lw | sw | beq | j | ori | Lui |
|---|---|---|---|---|---|---|---|
| | 000000 | 100011 | 101011 | 000100 | 000010 | 001101 | 001111 |
| MemWrite | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| RegDst | 1 | 0 | X | X | X | 0 | 0 |
| ALUSrc | 0 | 1 | 1 | 0 | X | X | X |
| MemtoReg | 0 | 1 | X | X | X | 0 | 0 |
| ALUop | 010 | 000 | 000 | 001 | XXX | 011 | 100 |
| branch | 0 | 0 | 0 | 1 | X | 0 | 0 |
| jump | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| EXT | 00 | 00 | 00 | 00 | XX | 01 | 10 |

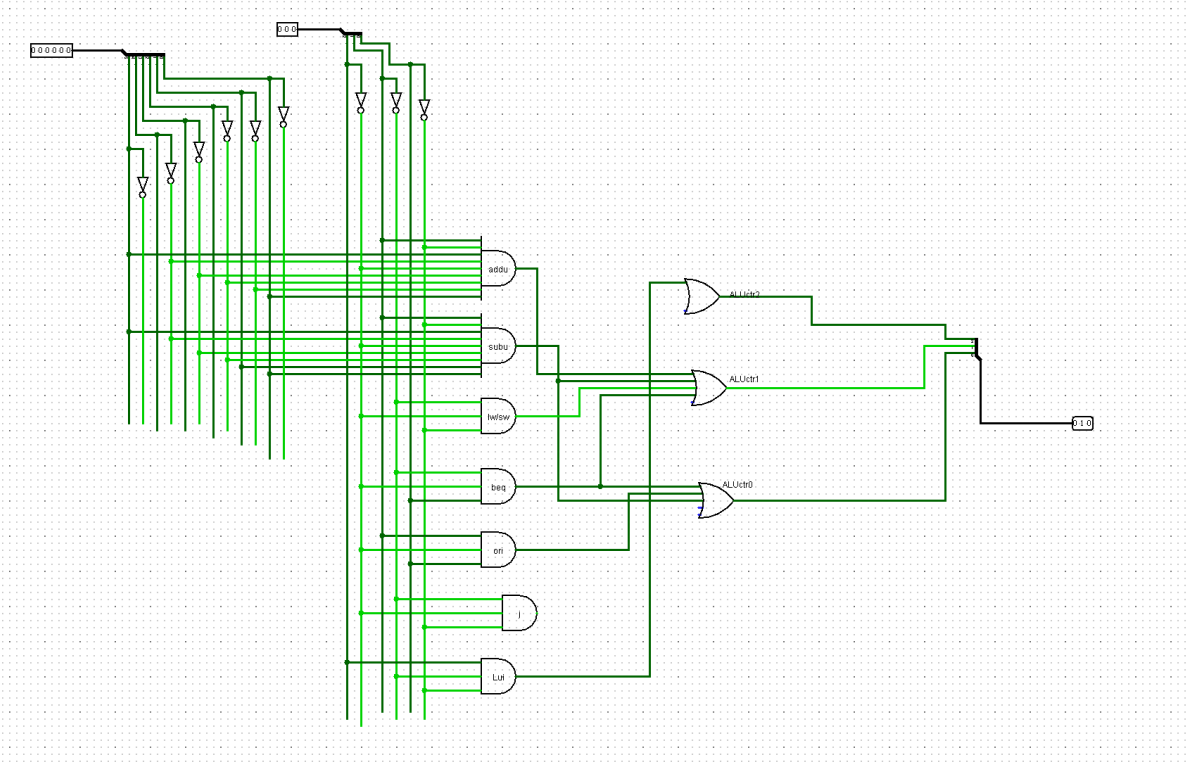



**ALUop编码**

| 编码 | 描述 |
| --- | --- |
| 000 | 加法 |
| 001 | 减法 |
| 010 | 依赖于func字段 |
| 011 | A \|B |
| 100 | A⊕B |

**ALU 译码器真值表**

| 指令名称 | func | ALUop | ALUctrl |
| --- | --- | --- | --- |
| lw | | 000 | 010 |
| sw | | 000 | 010 |
| beq | | 001 | 011 |
| j | | XXX | XXX |
| ori | | 011 | 001 |
| lui | | 100 | 100 |
| addu | 100001 | 010 | 010 |
| subu | 100011 | 010 | 011 |




# 二 测试方案

测试ori指令以及32个寄存器是否存在问题

```
.text
ori $0, $0, 0
ori $1, $1, 1
ori $2, $2, 2
ori $3, $3, 3
ori $4, $4, 4
ori $5, $5, 5
ori $6, $6, 6
ori $7, $7, 7
ori $8, $8, 8
ori $9, $9, 9
ori $10, $10, 10
```

```
ori $11, $11, 11
ori $12, $12, 12
ori $13, $13, 13
ori $14, $14, 14
ori $15, $15, 15
ori $16, $16, 16
ori $17, $17, 17
ori $18, $18, 18
ori $19, $19, 19
ori $20, $20, 20
ori $21, $21, 21
ori $22, $22, 22
ori $23, $23, 23
ori $24, $24, 24
ori $25, $25, 25
ori $26, $26, 26
ori $27, $27, 27
ori $28, $28, 28
ori $29, $29, 29
ori $30, $30, 30
ori $31, $31, 31
```

测试beq指令、addu、subu指令

```
.text
ori $t1, $t1, 1
ori $t2, $t2, 1
ori $t3, $t3, 3
reset:
beq $t1, $t2, jump
ori $t6, $t6, 6
ori $t7, $t7, 7
ori $t8, $t8, 8
jump:

addu $t4, $t1, $t2
addu $t5, $t2, $t3

beq $t1, $t3, reset
ori $s1, $s1, 6
ori $s2, $s2, 7
subu $s1 $s1,  $t1
subu  $s2, $s2, $t2
```

测试sw指令

```
.text
ori $1, $1, 1
ori $2, $2, 2
ori $3, $3, 3
ori $4, $4, 4
ori $5, $5, 5
```

```
ori $6, $6, 6
ori $7, $7, 7
ori $8, $8, 8
ori $9, $9, 9
ori $10, $10, 10
ori $11, $11, 11
ori $12, $12, 12
ori $13, $13, 13
ori $14, $14, 14
ori $15, $15, 15
sw $1, 4($0)
sw $2, 8($0)
sw $3, 12($0)
sw $4, 16($0)
sw $5, 20($0)
sw $6, 24($0)
sw $7, 28($0)
sw $8, 32($0)
sw $9, 36($0)
sw $10, 40($0)
sw $11, 44($0)
sw $12, 48($0)
sw $13, 52($0)
sw $14, 56($0)
sw $15, 60($0)
```

测试 Lui指令以及j指令

```
.test
ori $1, $1, 1
ori $2, $2, 2
lui $3, 0x3039
lui $4, 0x6011
ori $5, $5, 5
ori $6, $6, 6
test:
addu $7, $1, $2
addu $8  $5, $6
j test
ori $1, $1, 1
ori $2, $2, 2
```

测试lw指令

```
.text
ori $1, $1, 1
ori $2, $2, 2
ori $3, $3, 3
ori $4, $4, 4
ori $5, $5, 5
ori $6, $6, 6
ori $7, $7, 7
ori $8, $8, 8
sw $1, 96($0)
sw $2, 100($0)
sw $3, 104($0)
```

```
sw $4, 108($0)
sw $5, 112($0)
sw $6, 116($0)
sw $7, 120($0)
sw $8, 124($0)
lw $16, 96($0)
lw $17, 100($0)
lw $18, 104($0)
lw $19, 108($0)
lw $20, 112($0)
lw $21, 116($0)
lw $22, 120($0)
lw $23, 124($0)
```

# 三 思考题

### 1.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。

合理，IM的作用是存储指令，在我们搭建的CPU中，指令只需要在测试前写入，在执行指令的过程中，指令数据不能改变，不需要写入，而ROM是只读存储器，所以，IM用ROM是合理的。而储存器DM在执行指令的过程中需要对数据进行写入和读出，但是对速度要求不高，所以DM选用可读可写的RAM比较合理。GRF要经常对数据进行读写处理而且速度要求较高，所以，GRF使用register也很合理。

### 2.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。

nop空指令唯一的操作就是对下一PC取PC+4，nop对于数据通路没有任何影响，所以我们不需要将它加入控制信号真值表。而对于下一地址的计算，在不写入控制信号真值表的情况下，我们的多路选择器控制信号的默认值为0，可以保证下一PC是PC+4，所以能满足对下一地址的计算，不需要加入控制信号真值表。

### 3. 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。

假设MARS中DM的地址范围为0x2000-0x3ffff， 而设计的CPU中DM地址是以0x0000开始，所以实际储存在CPU DM中的地址为0x0000-0x1ffff ，那我们就要实现- 0x2000的操作，可以使用spliiter分线器，只选取后29位信号，那样0x2000-0x3000就能映射到0x0000-0x1fff

### 4. 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。

所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。

对组合逻辑来说，不存在有限状态机，其输出值不依赖于前面的输入值X。这时只要对每个输入向量证明其输出向量相同。

对一个时序电路而言，可以把它看成一个有限状态机。电路功能的等价可以用有限状态机的等价来判断。假定有两个状态机A和B，要对它们进行比较。直观的说，当A和B有相同的接口，而且从相同的初始状态出发，两者对有效输入值序列产生相同的输出值序列，则可以说A和B等价。

形式验证的优点如下：

(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了100%。
(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，不需要开发测试激励。
(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：

形式验证到现在还不能有效地验证电路的性能，比如电路的延时和能耗。