# 计算机组成CPU设计文档-P4

[计算机组成CPU设计文档-P4 1](#_Toc530612645)

[1、 CPU基本参数与指标 2](#_Toc530612646)

[2、 CPU模块结构 2](#_Toc530612647)

[一、 模块规格（数据通路） 3](#_Toc530612648)

[1、 数据通路 3](#_Toc530612649)

[2、 IFU（取指令单元） 3](#_Toc530612650)

[3、 GRF（通用寄存器组） 4](#_Toc530612651)

[4、 ALU（逻辑运算单元） 5](#_Toc530612652)

[5、 DM（数据存储器） 6](#_Toc530612653)

[6、 Ext（位数扩展器） 7](#_Toc530612654)

[7、 Nadd（分支指令Pc计算器） 7](#_Toc530612655)

[8、 Jump（绝对跳转指令Pc计算器） 7](#_Toc530612656)

[二、 模块规格（控制电路） 8](#_Toc530612657)

[三、 CPU功能测试 11](#_Toc530612658)

[1、 功能测试原则 11](#_Toc530612659)

[2、 测试策略 11](#_Toc530612660)

[3、 测试实例 11](#_Toc530612661)

[四、 本章思考题 17](#_Toc530612662)

[9. 有关CPU扩展的说明 20](#_Toc530612663)

整体结构与概览

## CPU基本参数与指标

处理器类型：单周期

处理器字长：32位

处理器支持指令集：{addu, subu, jr, sll(nop), ori, lui, beq, lw, sw, jal}

顶层封装模块端口：

表格 1 顶层封装模块端口

|  |  |  |
| --- | --- | --- |
| 端口名 | 类型 | 描述 |
| Reset | In | 接受同步复位信号，对PC、GRF和Mem复位。 |
| Clock | In | 接受时钟驱动信号，驱动PC、GRF/W、IM/W |

## CPU模块结构

### 数据通路

1、IFU（取指令单元）：包括PC和存放指令的ROM，用于输出当前指令码。

2、GRF（通用寄存器组）：内含32个寄存器，支持对寄存器值的读写。

3、ALU（算术逻辑单元）：运算执行部件，对32位数执行多种运算。

4、DM（数据存储器）：存储数据部件，支持读写。

5、EXT（位扩展器）：将16位数扩展为32位数，支持有/无符号扩展。

6、Nadd（条件分支计算器）：根据Pc+4和立即数，计算分支成立的Pc值。

7、Jump（无条件跳转Pc计算器）：根据Pc和立即数，计算Jal指令Pc值。

### 控制信号

1. 控制器：识别指令并生成CPU各部分的控制信号，使用逻辑阵列实现。

## 模块规格（数据通路）

## 数据通路

表格 2 数据通路端口总表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PC | IF | GRF | | | | ALU | | | DM | | Ex |
| Instr | PC | IF | RA1 | RA2 | WA | WD | A | B | C | Addr | WD | Ex-32 |
| Addu | Pc+4 | Pc | IF[25:21] | IF[20:16] | IF[15:11] | ALU | R1 | R2 |  |  |  |  |
| Subu | Pc+4 | Pc | IF[25:21] | IF[20:16] | IF[15:11] | ALU | R1 | R2 |  |  |  |  |
| Jr | R1 | Pc | IF[25:21] |  |  |  |  |  |  |  |  |  |
| Sll | Pc+4 | Pc |  | IF[20:16] | IF[15:11] | ALU |  | R2 | IF[10:6] |  |  |  |
| Ori | Pc+4 | Pc | IF[25:21] |  | IF[20:16] | ALU | R1 | ZE(IM) |  |  |  | IF[15:0] |
| Lui | Pc+4 | Pc |  |  | IF[20:16] | ALU |  | ZE(IM) |  |  |  | IF[15:0] |
| Beq | Nadd|Pc4 | Pc | IF[25:21] | IF[20:16] |  |  | R1 | R2 |  |  |  |  |
| Lw | Pc+4 | Pc | IF[25:21] |  | IF[20:16] | DM | R1 | SE(IM) |  | ALU |  | IF[15:0] |
| Sw | Pc+4 | Pc | IF[25:21] | IF[20:16] |  |  | R1 | SE(IM) |  | ALU | R2 | IF[15:0] |
| Jal | Jump | Pc |  |  | $31 | ALU |  |  |  |  |  |  |

\*SE : Sign-Extend; ZE: Zero-Extend

\*\*Lui指令操作与指令手册存在细微差异，为统一化预先进行了32位扩展。

\*\*\*下划线端口：存在多路数据输入，需要使用选择器。

## IFU（取指令单元）

### 端口

表格 3 IFU端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受Pc同步复位信号 |
| Branch\_Jump | In | 控制信号，接受指令是否为跳转指令 |
| Pc\_Update[31:0] | In | 数据通路，分支/跳转指令中接受PC更新值 |
| PC[31:0] | Out | 数据通路，输出PC（32位**Byte编址**） |
| Instr[31:0] | Out | 数据通路，输出32位指令二进制码 |

### 功能描述

* IFU主要由PC和存放指令的ROM组成，用于取出指令和PC更新。
* ROM规格为1024\*32bits，**字编址（访问时需地址转换）**。
* PC为32位二进制，起始地址为0x00003000，**字节编址**，支持向字编址转换（除4）。

### 注意事项

* ROM和RAM部件的地址端口为字编址地址。为保证兼容性，该设计在顶层设计时用字节编址，而在次层具体部件设计时会进行字编址转换。

## GRF（通用寄存器组）

### 端口

表格 4 GRF端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受同步复位信号 |
| RegWrite | In | 控制信号，接受寄存器写使能信号 |
| ReadAddr1/R[4:0] | In | 数据通路，读，接受Rs寄存器地址 |
| ReadAddr2/R[4:0] | In | 数据通路，读，接受Rt寄存器地址 |
| WriteAddr/W[4:0] | In | 数据通路，写，接受被写入寄存器地址 |
| WriteData/W[31:0] | In | 数据通路，写，接受被写入数据 |
| RegData1/R[31:0] | Out | 数据通路，读，输出Rs寄存器值 |
| RegData2/R[31:0] | Out | 数据通路，读，输出Rt寄存器值 |
| WPC[31:0] | None | 调试信号，用于寄存器写时display |

### 功能描述

* GRF中共有32个寄存器，对应MARS中的32个通用寄存器。（注意：不包括hi，lo，pc寄存器。）
* 读：GRF读功能时作为组合逻辑电路，根据输入地址信号，输出数据。
* 写：GRF写功能作为时序逻辑电路，相应的Addr, Data, RegWrite应在时钟上升沿前做好准备。

### 备注

* 此版本GRF读写功能的地址和数据端口独立，可实现同步读写操作。
* 0号寄存器恒为0值，不可被改写。

## ALU（逻辑运算单元）

### 端口

表格 5 ALU端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| AluOp[3:0] | In | 控制信号，接受算术逻辑信号 |
| A[31:0] | In | 数据通路，接受算术逻辑操作数A |
| B[31:0] | In | 数据通路，接受算术逻辑操作数B |
| C[4:0] | In | 数据通路，接受算术逻辑操作数C（常于移位） |
| Result[31:0] | Out | 数据通路，输出结果 |

### 功能描述

* ALU受AluController的控制信号控制，输出不同的算术逻辑结果：

表格 6 ALU功能表

|  |  |  |
| --- | --- | --- |
| AluOp | 功能 | 适用指令 |
| 000/0 | 32位 +，不带溢出检测 | addu, lw, sw |
| 001/1 | 32位 - ，不带溢出检测 | subu |
| 010/2 | 32位 | | ori |
| 011/3 | 32位 compare = | beq |
| 100/4 | 32位 <<16 | lui |
| 101/5 | 32位 =A | jr |
| 110/6 | 32位 左移运算B<<C | sll(nop) |

## DM（数据存储器）

### 端口

表格 7 数据存储器端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受异步复位信号 |
| MemWrite | In | 控制信号，接受写使能信号 |
| MemRead | In | 控制信号，接受读使能信号 |
| MemAddr[31:0] | In | 数据通路，接受**读/写**操作地址, **byte编址** |
| WriteData[31:0] | In | 数据通路，写，写入数据 |
| ReadData[31:0] | Out | 数据通路，读，输出数据 |
| WPC[31:0] | None | 调试信号，用于主存写时display |

### 描述

* 数据存储器使用RAM实现，容量为1024\*32bits，RAM地址为**字编址**。
* 读/写共用一个地址端口，同一时钟周期只能进行读/写的其中之一。
* 起始地址：0x00000000。

## 

## Ext（位数扩展器）

表格 8 16-32位扩展器端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| ExtOp | In | 控制信号，控制扩展方式（0-zero，1-sign） |
| In[15:0] | In | 数据通路，接收待扩展的16位数字。 |
| Out[31:0] | Out | 数据通路，输出扩展后的32位数字。 |

## Nadd（分支指令Pc计算器）

表格 9 Nadd端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| PC+4[31:0] | In | 数据通路，不发生跳转时下条指令地址 |
| Offset[31:0] | In | 数据通路，偏移量（**正负，字编址**） |
| Out[31:0] | Out | 数据通路，PC + 4 + Offset << 2 |

## Jump（绝对跳转指令Pc计算器）

表格 10 Jump端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| PcHead[3:0] | In | 数据通路，Pc大区编址 |
| Jim[25:0] | In | 数据通路，J/Jal指令立即数（**绝对字编址**） |
| Out[31:0] | Out | 数据通路，{PcHead, Jim, 00} |

## 模块规格（控制电路）

本设计电路中，由于Verilog条件判断的易操作性和jr型指令与其他R型指令在控制信号上存在许多差异。因此不同于Logisim中使用Controller和ALUController 分别控制，本电路仅采用Controller进行控制。

### Controller 主控单元

* 1. 端口

表格 11 主控单元端口功能表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能（所在通路，作用部件，描述） |
| Op | In | 数据通路，指令的Instr[31:26] |
| Func | In | 数据通路，指令的Instr[5:0] |
| RegWrite | Out | 控制信号，GRF，GRF写使能信号 （1–允许，0-不允许） |
| MemRead | Out | 控制信号，Mem，Mem读使能信号（1-允许，0-不允许:高阻） |
| MemWrite | Out | 控制信号，Mem，Mem写使能信号（1-允许，0-不允许） |
| Branch\_Jump | Out | 控制信号，IF，IF接受外部Pc更新信号（1-允许，0-不允许） |
| WaSel[1:0] | Out | 控制信号，MUX，GRF写地址选择  0：Rt  1：Rd  2：$31(ra) |
| WdSel[1:0] | Out | 控制信号，MUX，GRF写数据选择  0：Alu  1：Memory  2：Pc |
| ExtOp | Out | 控制信号，16-32位扩展类型（1-符号扩展，0-无符号扩展） |
| AluSrc | Out | 控制信号，MUX，ALU-B端口选择器信号（1-扩展器，0-RD2） |
| AluOp[3:0] | Out | 控制信号，ALU，ALU驱动信号（具体请参加ALU功能表） |
| nPc\_Sel[1:0] | Out | 控制信号，MUX，Pc更新值选址  0：Branch部件  1：Jump部件  2：Alu部件（jr） |

* 1. 信号真值表

表格 12 理论信号真值表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | addu | Subu | Jr | Sll | Ori | Lw | Sw | Beq | Lui | Jal |
| Op | 000000 | 000000 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 |
| Func | 100001 | 100011 | 001000 | 000000 | xxxxxx | xxxxxx | xxxxxx | xxxxxx | xxxxxx | xxxxxx |
| RegWrite | 1 | 1 | x | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| MemRead | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Branch\_Jump | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| WaSel[1:0] | 1 | 1 | xx | 1 | 0 | 0 | xx | xx | 0 | 2 |
| WdSel[1:0] | 0 | 0 | xx | 0 | 0 | 1 | xx | xx | 0 | 2 |
| ExtOp | x | x | x | x | 0 | 1 | 1 | 1 | x | x |
| AluSrc | 0 | 0 | x | 0 | 1 | 1 | 1 | 0 | 1 | x |
| AluOp[3:0] | 0 | 1 | 5 | 6 | 2 | 0 | 0 | 3 | 4 | xxxx |
| nPc\_Sel[1:0] | xx | xx | 2 | xx | xx | xx | xx | 0 | xx | 1 |

表格 13 实现信号真值表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | addu | Subu | Jr | Sll | Ori | Lw | Sw | Beq | Lui | Jal |
| Op | 000000 | 000000 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 |
| Func | 100001 | 100011 | 001000 | 000000 | xxxxxx | xxxxxx | xxxxxx | xxxxxx | xxxxxx | xxxxxx |
| RegWrite | 1 | 1 | 0(x) | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| MemRead | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Branch\_Jump | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| WaSel[1:0] | 1 | 1 | 0(xx) | 1 | 0 | 0 | 0(xx) | 0(xx) | 0 | 2 |
| WdSel[1:0] | 0 | 0 | 0(xx) | 0 | 0 | 1 | 0(xx) | 0(xx) | 0 | 2 |
| ExtOp | 0(x) | 0(x) | 0(x) | 0(x) | 0 | 1 | 1 | 1 | 0(x) | 0(x) |
| AluSrc | 0 | 0 | 0(x) | 0 | 1 | 1 | 1 | 0 | 1 | 0(x) |
| AluOp[3:0] | 0 | 1 | 5 | 6 | 2 | 0 | 0 | 3 | 4 | 0(xxxx) |
| nPc\_Sel[1:0] | 0(xx) | 0(xx) | 2 | 0(xx) | 0(xx) | 0(xx) | 0(xx) | 0 | 0(xx) | 1 |

\*: beq指令：进入ALU比较的值都源于寄存器；立即数为符号扩展。

\*\*：lui指令与ori指令：前者完全不顾及GRF读操作，且对扩展方式不在意。

## CPU功能测试

## 功能测试原则

* 所测试的指令不能够超出CPU支持的范围。（谨防同指令标识的拓展指令。）
* 测试的首要目标：全面（例如：正负性、边界数据、边界存储等）。
* 测试步骤：控制信号 -> 单指令 -> 多指令，寄存器、主存 -> 综合

## 测试策略

1. 控制信号正确性检查：独立检查每条指令的控制信号是否符合设计。
2. 单指令正确性检查：
   1. Addu, subu: 正正（溢出/不溢），负负（溢出/不溢），正负，负正。
   2. Lui，ori：负数的构造。
   3. Sw，lw：非零数边界存储，正负偏移存储。
   4. Beq：成立与不成立跳转，下跳与上跳。
   5. Sll检查：Nop功能。
   6. Jal检查：正负绝对跳转，$31赋值细节检查。（pc4）
   7. Jr检查：$31, $0, $8检查
3. 综合性程序检查：带条件的存值+带条件的取值。

## 测试实例

### 控制信号正确性检查

使用下列代码检查各指令运行时，控制信号值是否符合期望，控制信号期望输出于第三节以表格给出，在此不再赘述。

addu $t1, $t2, $t3

subu $t1, $t2, $t3

lw $t1, 12($t2)

sw $t1, 12($t2)

ori $t2, $t2, 0xffff

lui $t1, 0xf

sll $t1, $t2, 2

back:

beq $t1, $t2,back

jal next

nop

next:

jr $ra

jr $t0

代码 1 控制信号正确性检查代码

### 单指令正确性检查

单指令正确性检查首先是lui和ori，因为这两条指令才能构造非零数。而后，利用构造完成的各种数值，对指令addu、subu、lw和sw进行测试。

测试代码与期望结果如下：

# 单指令运算正确性检查:addu, subu, lw, sw, ori, lui   # 检查内容与期望结果

# Lui, Ori

lui $s0, 0x7ff1                 # 功能检查：$16 = 0x7ff10000

ori $s1, $zero, 0xf001          # 功能检查：$17 = 0x0000f001

lui $s2, 0xffff

ori $s3, $s2, 0xfffc            # 负数构造：$19 = -4

# addu, subu

addu $t0, $s0, $s1              # 不溢出相加：$8 = 0x7ff1f001

addu $t1, $s0, $s0              # 溢出相加：$9 = 0xffe20000

addu $t2, $s3, $s3              # 负负相加 $10 = -8

addu $t3, $s1, $t2              # 正负相加 $11 = 61433

subu $t4, $s1, $t3              # 正正相减 $12 = 8

subu $t5, $t3, $s1              # 正正相减 $13 = -8

subu $t6, $s3, $t5              # 负负相减 $14 = 4

lui $s4, 0x8000

subu $t7, $s4, $t4              # 负正相减(溢出) $15 = 0x7ffffff8

# sw, lw

ori $t8, 0

sw $t0, 0($t8)                  # sw下边界测试：mem[0] = 0x7ff1f001

sw $t1, 16($t8)                 # sw 偏移量测试：mem[4] = 0xffe20000

ori $t9, 4092

sw $t4, 0($t9)                  # sw上边界测试：mem[1023] = 8

sw $t5, -16($t9)                # sw 负偏移测试：mem[1019] = -8

ori $t8, 13

lw $s4, 3($t8)                  # 和为4倍偏测试：$20 = 0xffe20000

lw $s5, 0($t9)                  # 上边界测试 $21 = 8

lw $s6, 4($t9)                  # 无效界读入测试

代码 2 单指令真确性检查代码1

而后针对beq、jal、jr指令进行检测

# 单指令正确性检查：ori, lui, beq, jal, jr       # 检查期望结果和pc跳转情况

# beq                       # 检测目的与预期结果

ori $t0, 1

ori $t1, 2

ori $t2, 3

ori $t4, 1

addu $s0, $s1, $s2              # 非跳转语句选择无效检测：pc = pc + 4 (not pc + 4 + 0x8021)

beq $t0, $t1, label2                # 不跳转检测：pc = pc+4

nop

label1:

    lui $s0, 0xf000

    beq $t0, $t4, label4            # 正跳转检测：pc = pc+4+8

label2:

    lui $s1, 0x0f00

label3:

    lui $s2, 0x00f0

    jal function\_begin

    ori, $t5, 0x3048                # 非ra跳转，至return标签

    jr $t5

label4:

    lui $s3, 0x000f

    addu $t0, $t0, $t4

    beq $t0, $t1, label3            # 逆跳转检测：pc = pc+4-16

    jal function\_end

return:

    ori $s6, $zero, 0x00f0

    jal function\_end

function\_begin:

    ori $s4, $zero, 0xf000

    jr $ra

    ori $s5, $zero, 0x0f00

function\_end:

# 顺序：label1 - > label4 -> label 3-> function\_begin -> return -> function\_end

# 最终期望输出：$16 = 0xf0000000, $17 = 0, $18 = 0x00f00000, $19 = 0x000f0000, $20 = 0x0000f000, $21 = 0, $22 = 000000f0, $31 = 00003050, $8 = 3

代码 3 单指令正确性检查代码2

### Beq，jal，jr检查：死循环构造

# beq，jal，jr测试：死循环维护

ori $t0, 100

ori $t1, 50

ori $t2, 150

addu $t1, $t1, $t0

label:

    beq $t1, $t2, loop1

    jal end

loop1:

    jal loop2

    jal label

loop2:

    jr $ra

end:

### 综合性程序检查：递归运算检查。检查时，主要检查输出运行步骤和$16的值是否和Mars一致。

# p4指令综合测试

# addu, subu, jr, sll(nop), ori, lui, lw, sw, beq, jal

ori $t1, 4

ori $t2, 4                  # 4层级别的递归

ori $t3, 1

lui $sp, 0

ori $s7, 0x3044

ori $sp, 4096                   # initial $sp to 4092

ori $a0, 1                  # set 调用函数参数

jal function\_begin

nop

jal end

function\_begin:

    subu $sp, $sp, $t1

    sw $s0, 0($sp)

    subu $sp, $sp, $t1

    sw $ra, 0($sp)

    beq $a0, $t2, return            # if(a0 == t2)return;

    lui $s0, 0              # li $s0 0

    ori $s0, 0              # int i = 1;

    loop:

        # start of main content

        addu $s1, $s1, $s0      # s1 += i;

        subu $sp, $sp, $t1      # backup

        sw $a0, 0($sp)

        addu $a0, $a0, $t3      # a0 ++

        jal function\_begin      # call next layer

        nop

        lw $a0, 0($sp)

        addu $sp, $sp, $t1      # recover

        # end of main content

        addu $s0, $s0, $t3      # i++

        beq $s0, $a0, loop\_end

        nop

        jr $s7              # 此语句在程序中的编号不可以发生变化

    loop\_end:

    return:

    lw $ra, 0($sp)

    addu $sp, $sp, $t1

    lw $s0, 0($sp)

    addu $sp, $sp, $t1

    jr $ra

    nop

function\_end:

end:

# 期望输出：$16 = 0 + 1\*(0 + 1) +...+(n-2)\*(0 + 1 + ... + n-2), 当n=4时，结果为7

代码 4 综合性功能检测代码

## 本章思考题

1. 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**：在使用reg模拟的DM中，reg的大小实际上为32位即一字大小，而输入的地址是按照byte编址的，为了转换为word编址，则需要右移两位，也便等价于选择[11:2]这个区段。这个addr信号来源于alu计算和取位的结果。

1. 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信号对象时GRF、PC和DM。复位可以理解为重新执行这一段程序，因此要对电路中“有长久影响”的“时序部件”进行恢复，这其中便包括PC、GRF和DM（IM由于存储指令固不进行复位）。

1. 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：**

表格 14 编码方式表

|  |  |
| --- | --- |
| 编码方式 | 样例 |
| Assign+三目赋值运算 | Assign Rtype = (Op = 000000) ? 1:0; |
| Case语句 | Case(Op):  6’b000000: {Rtype, lw} = 2’b10;  6’b100011:{Rtype, lw} = 2’b01;  Default: {Rtype, lw} = 2’b00; |
| If语句 | If(Op == 6’b000000)  {Rtype, lw} = 2’b10;  Elseif(Op == 6’b100011)  {Rtype, lw} = 2’b01;  Else {Rtype, lw} = 2’b00; |

1. 根据你所列举的编码方式，说明他们的优缺点。

**答**：

* 1. assign语句+三目运算符。优点：极致简洁，无需考虑锁存器问题。缺点：在条件较多时，需要嵌套，容易出现错误。
  2. case语句。优点：简洁，接口性强。缺点：多变量赋值时容易造成锁存器，只能判断单一变量。
  3. if-else语句。优点：灵活性强，C语言形式。缺点：多变量赋值时易造成锁存器，新增维护时不方便。

1. 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**：addi、add之于addiu、addu的差异是33位\32位操作和在利用加法器计算出结果后是否需要进行溢出判断。而带溢出判断的指令之所以将加法操作数变为33位其原理是分析“次高位和最高位是否有相同的进位”以判断是否溢出，最后也会只取出低32位的数（等价于32位操作数运算）。因此若始终忽略溢出语句，那么if（true）的内容可以直接被忽视，易知是等价的。

1. 根据自己的设计说明单周期处理器的优缺点。
   1. 优点
      1. 以组合逻辑电路为主，时序部件很少，电路简单易实现。
      2. 指令执行完全按照顺序，不存在依赖（结构、数据、控制）的问题。
   2. 缺点
      1. 运行时间慢，时钟周期取决于最慢执行指令的时常。
      2. 低吞吐率。
2. 简要说明jal、jr和堆栈的关系。

**答**：jal、jr和堆栈三者常常一并使用以完成子函数的调用功能，三者运用在子函数调用时往往会按照：寄存器备份入栈-jal跳转记录-函数执行-寄存器恢复出栈-jr返回调用位置的顺序执行。因此也可以理解为jal与堆栈的压入相关，而jr与堆栈的弹出相关。

## 有关CPU扩展的说明

本单周期CPU支持一定的扩展，允许使用Logisim软件对线路进行调整，以完成更多指令功能，用户在自行拓展时，应主要遵循以下几个步骤。

* 分析指令，查阅现有Controller真值表和数据通路总表。
* 手工绘制新增指令的数据通路，并写出需要修改和新增的控制信号。（考虑是否新增数据元件）
* 先完善数据通路。先微观：以每个功能子电路为单位新增和修改功能；后宏观：根据端口表修改通路。
* 后完善控制电路：根据分析结果在Controller中新增或修改控制信号。
* 测试阶段：控制信号测试-单指令多情况测试-综合测试。

\*：在现场测试修改时，**若对v文件进行了改动，请务必在文档文件起头注释。**