# 北航2021计组P4设计文档

**黄雨石20376156**

# 一、CPU设计方案

## (一)总体设计概述

使用 `Verilog` 开发一个简单的单周期 `CPU`，总体概述如下：

1. 此 `CPU` 为 `32` 位 `CPU`
2. 此 `CPU` 为单周期设计
3. 此`CPU`支持的指令集为：`{addu, subu, ori, lw, sw, beq, lui,jal,jr nop}`
4. `nop` 机器码为`0x00000000`
5. `addu`，`subu`不支持溢出

## (二)关键模块定义

1. **ALU**

   **ALU端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| SrcA[31:0] | I | 数据1 |
| SrcB[31:0] | I | 数据2 |
| Result[31:0] | O | 输出的数据 |
| ALUControl[3:0] | I | 控制信号 000：加 001：减 010：或 011：比较是否相等 100：左移16位 |
| ALUZero | O | 判断A与B是否相等 |
| shamt[4:0] | I | 移位数 |

   **ALU功能描述**

   1. 加：A+B
   2. 减：A-B
   3. 或：A|B
   4. 是否相等：A==B?
   5. 左移16位：A<<16

2. **IFU**

   **一、PC**

   **PC端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| NPC[31:0] | I | 32位输入地址 |
| PC[31:0] | O | 32位输出地址 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |

## 二、NPC

**NPC端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| PC[31:0] | I | 32位输入地址 |
| im26[25:0] | I | 26位立即数 |
| im16[15:0] | I | 16位立即数 |
| NPC[31:0] | O | 32位输出地址 |
| ALUZero | I | 判断A与B是否相等（判断beq） |
| NPCSel[2:0] | I | 控制信号 00：加4 01：beq指令 |
| RD1[31:0] | I | rs寄存器的值 |
| PC4 | O | PC+4 |

## 三、IM

**IM端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| PC[31:0] | I | 指令地址 |
| instr[31:0] | O | 32位指令 |
| rs[4:0] | O | rs地址 |
| rt[4:0] | O | rt地址 |
| rd[4:0] | O | rd地址 |
| im16[15:0] | O | 16位立即数 |
| im26[25:0] | O | 26位立即数 |
| shamt[4:0] | O | 5位移位数 |

**IFU功能描述**

1. 复位 复位信号有效时，PC被设置为0x0000_3000

2. 取指令 根据PC当前值从IM中取出指令。
3. 计算下一条指令地址。如果当前指令不是beq,jal,jr指令，PC=PC+4；如果当前指令是beq指令且 zero=1，则PC=PC+4+sign_extend(offset||02); 如果当前指令是jal或者j指令那么 PC = {PC[31:28],im26,2'b00}；如果当前指令是jr那么 PC = RD1

3. **Ctrl**

**Ctrl端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| instr[31:0] | I | 32位指令信号 |
| NPCSel[2:0] | O | NPC控制信号 |
| ExtOp | O | Ext控制信号 |
| GWDSel[1:0] | O | GRF写入数据控制信号 |
| A3Sel[1:0] | O | GRF写入地址控制信号 |
| GWE | O | GRF写入使能控制信号 |
| SrcBSel[1:0] | O | ALU的SrcB控制信号 |
| ALUControl[3:0] | O | ALU控制信号 |
| DWE | O | DM写入使能控制信号 |
| DMSel | O | DM控制信号 |

**Ctrl功能描述**

| 端口 | addu | subu | ori | lw | sw | lui | beq | jr | jal |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 00111 | 000100 | 001000 | 000011 |
| func | 100001 | 100011 | | | | | | | |
| NPCSel[2:0] | 000 | 000 | 000 | 000 | 000 | 000 | 001 | 011 | 010 |
| ExtOp | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| GWDSel[1:0] | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 00 | 10 |
| A3Sel[1:0] | 00 | 00 | 01 | 01 | 00 | 01 | 00 | 00 | 10 |
| GWE | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| SrcBSel | 00 | 00 | 01 | 01 | 01 | 01 | 00 | 00 | 00 |
| ALUControl[3:0] | 0000 | 0001 | 0010 | 0000 | 0000 | 0100 | 0011 | 0000 | 0000 |
| DWE | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| DMSel | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |

4. **Ext**

**Ext端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| im16[15:0] | I | 代扩展的 16 位信号 |
| ExtOut[31:0] | O | 扩展后的 32 位的信号 |
| ExtOp | I | 无符号或符号扩展选择信号 0：无符号扩展 1：符号扩展 |

**Ext功能描述**

1. 当 ExtOp为 0 时，将 imm16 无符号扩展输出
2. 当 ExtOp为 1 时，将 imm16 符号扩展输出

5. **GRF**

**GRF端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| GWD[31:0] | I | 写入A3对应寄存器的数据 |
| RD1[31:0] | O | A1地址对应寄存器数值 |
| RD2[31:0] | O | A2地址对应寄存器数值 |
| A1[4:0] | I | RD1中数据的寄存器地址 |
| A2[4:0] | I | RD2中数据的寄存器地址 |
| A3[4:0] | I | 写入数据的寄存器地址 |
| GWE | I | 写使能信号 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| PC[31:0] | I | 指令地址 |

**GRF功能描述**

1. 复位。复位信号有效时所有寄存器的数值被设置为0x00000000
2. 读寄存器 根据当前输入的地址信号读出32位数据
3. 写寄存器。根据当前输入的地址信号，把输入的数据写入相应寄存器

6. **DM**

**DM端口说明**

| 端口名 | 方向 | 描述 |
| --- | --- | --- |
| Addr[4:0] | I | 数据的地址信号 |
| RD2[31:0] | I | 当WE为高电平时，写入地址A的数据 |
| DMOut[31:0] | O | 读出地址A的数据 |
| DWE | I | 高电平时允许写入数据 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| PC[31:0] | I | 指令地址 |
| DMSel[2:0] | I | DM控制信号 |

**DM功能描述**

1. 复位：复位信号有效时，所有数据被设置为0x00000000
2. 读：根据当前输入的寄存器地址读出数据
3. 写数据：根据输入地址，写入输入的数据

# 二、测试方案

**(1)测试代码**

mips

```
.text
ori $1,11
ori $2,22
ori $3,33
lui $4,12
lui $5,23
lui $6,24
lui $7,25
lui $8,34
lui $9,12
addu $10,$9,$9
addu $11,$2,$3
addu $12,$5,$6
subu $13,$3,$5
subu $14,$5,$4
subu $15,$2,$6
nop
lui $16,12
beq $9,$16, next #应跳转
nop
lui $1,1
lui $2,1
lui $3,1
lui $4,1
haha:
lui $5,1
lui $6,1
lui $7,1
```

```
lui $8,1
lui $9,1
lui $10,1
next:
beq $1,$2,haha #应不跳转，否则死循环
sw $1,0($0)
sw $2,4($0)
sw $3,8($0)
sw $4,12($0)
sw $5,16($0)
sw $6,20($0)
sw $7,24($0)
lw $17,0($0)
lw $18,4($0)
jal ok
lw $19, 8($0)
jal end
ok:
lw $0,0($0)
jr $31
end:
subu $3,$3,$0
subu $31,$0, $31
```

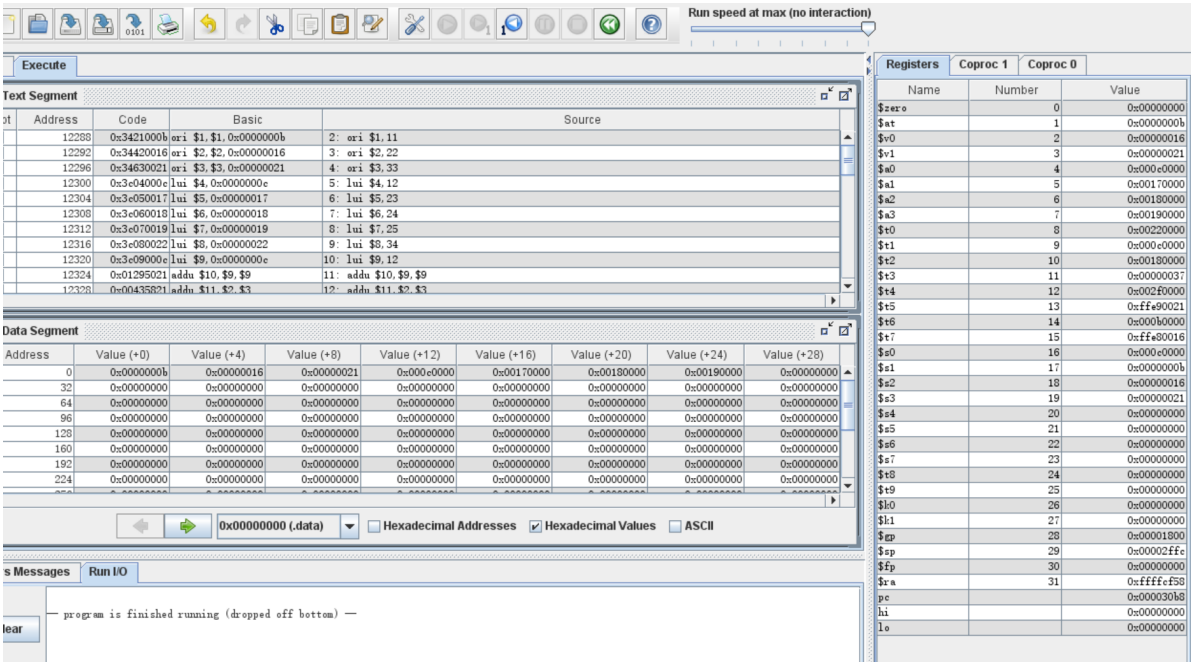
机器码

```
3421000b
34420016
34630021
3c04000c
3c050017
3c060018
3c070019
3c080022
3c09000c
01295021
00435821
00a66021
00656823
00a47023
00467823
00000000
3c10000c
1130000b
00000000
3c010001
3c020001
3c030001
3c040001
3c050001
3c060001
3c070001
3c080001
3c090001
3c0a0001
1022fff9
ac010000
```

```
ac020004
ac030008
ac04000c
ac050010
ac060014
ac070018
8c110000
8c120004
0c000c2a
8c130008
0c000c2c
8c000000
03e00008
00601823
001ff823
```

**(2)模拟结果**

mars




ise

```
@00003000: $ 1 <= 0000000b
@00003004: $ 2 <= 00000016
@00003008: $ 3 <= 00000021
@0000300c: $ 4 <= 000c0000
@00003010: $ 5 <= 00170000
@00003014: $ 6 <= 00180000
@00003018: $ 7 <= 00190000
@0000301c: $ 8 <= 00220000
@00003020: $ 9 <= 000c0000
@00003024: $10 <= 00180000
@00003028: $11 <= 00000037
@0000302c: $12 <= 002f0000
@00003030: $13 <= ffe90021
@00003034: $14 <= 000b0000
@00003038: $15 <= ffe80016
@00003040: $16 <= 000c0000
```

```
@00003078: *00000000 <= 0000000b
@0000307c: *00000004 <= 00000016
@00003080: *00000008 <= 00000021
@00003084: *0000000c <= 000c0000
@00003088: *00000010 <= 00170000
@0000308c: *00000014 <= 00180000
@00003090: *00000018 <= 00190000
@00003094: $17 <= 0000000b
@00003098: $18 <= 00000016
@0000309c: $31 <= 000030a0
@000030a8: $ 0 <= 0000000b  // 向0寄存器的写入会被测评机忽略，输出亦无妨
@000030a0: $19 <= 00000021
@000030a4: $31 <= 000030a8
@000030b0: $ 3 <= 00000021
@000030b4: $31 <= ffffcf58
```

二者结果一致

# 三、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？**

| 文件 | 模块接口定义 |
| --- | --- |
| dm.v | dm(clk,reset,MemWrite,addr,din,dout);<br>input clk; //clock<br>input reset; //reset<br>input MemWrite; //memory write enable<br>input [11:2] addr; //memory's address for write<br>input [31:0] din; //write data<br>output [31:0] dout; //read data |

`MIPS`中以字节为单位，而在我们设计的`DM`中，每一个`reg[31:0]`为一个单位。`Addr`来自`ALU`的输出端口，代表要读取的`DM`存储器的地址，在我们的`4KB`的`DM`设计中应当取`[11:0]`，又因为按字节寻址，因此取`[11:2]`

1. **思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。**

```
assign ALUOp = (addu | lw | sw) ? `ALU_add :
               (subu) ? `ALU_sub :
               (ori) ? `ALU_or :
               (lui) ? `ALU_lui :
               4'b0000;
```

```
case(ALUOp)
    `ALU_add: C = A + B;
    `ALU_sub: C = A - B;
    `ALU_or: C = A | B;
    `ALU_and: C = A & B;
    `ALU_xor: C = A ^ B;
    `ALU_sll: C = B << shamt;
    `ALU_srl: C = B >> shamt;
```

```
    `ALU_sra: C = $signed($signed(B) >> shamt);
    `ALU_lui: C = B << 16;

    // Ready for add other instructions...

    default: C = 32'h0000_0000;
endcase
```

此外还可以用 `if...else...` 语句

- `assign` 三目运算符不需要自己再定义寄存器
- `case` 语句和 `assign` 都可以通过宏定义的方式使代码更加美观，增强可读性
- `if...else...` 语句没用过

2. **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所驱动的部件具有什么共同特点?**

   都是存储器，例如 `PC` 、 `GRF` 和 `DM`

3. **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 。**

   根据 `RTL` 语言描述： `addi` 与 `addiu` 的区别在于当出现溢出时， `addiu` 忽略溢出，并将溢出的最高位舍弃； `addi` 会报告 `SignalException(IntegerOverflow)`

   `add` 于 `addu` 同理

4. **根据自己的设计说明单周期处理器的优缺点。**

   - 优点：设计简单，扩展性好
   - 缺点：时钟频率取决于执行时间最长的指令，整体时钟周期长，效率较低