

CPU设计方案综述

总体设计概述

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

关键模块定义

1.IFU

介绍

取指令单元，内部包括 PC(程序计数器)、IM(指令存储器)及相关逻辑，其中 PC 具有异步复位功能，IM 的容量为 容量为 32bit * 32，起始地址为 0x00000000。

端口定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|-------|-------|----|-------------|
| PCin | I | 32 | 设置下一个 PC 值。 |
| Clk | I | 1 | 时钟信号。 |
| Reset | I | 1 | 异步复位信号。 |
| Instr | O | 32 | 当前指令。 |
| PCout | O | 32 | 当前 PC 值。 |

功能定义

| 序号 | 功能名称 | 功能描述 |
|----|-------|---------------------------------|
| 1 | 异步复位 | 当异步复位信号有效时，将 PC 设置为 0x00000000。 |
| 2 | 取指令 | 当时钟上升沿到来时，Instr 输出当前 PC 对应的指令。 |
| 3 | 取PC值 | 当时钟上升沿到来时，PCout 输出当前 PC 值。 |
| 4 | 设置PC值 | 当时钟上升沿到来时，将 PCin 设置为当前 PC 值。 |

2.InstrSplitter

介绍

将 32 位指令划分为数个信号并输出。

端口与功能定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|-------|-------|----|---------------|
| Instr | I | 32 | 当前指令。 |
| Op | O | 6 | Instr[31:26]。 |
| Rs | O | 5 | Instr[25:21]。 |
| Rt | O | 5 | Instr[16:20]。 |
| Rd | O | 5 | Instr[11:15]。 |
| Func | O | 6 | Instr[5:0]。 |
| imm16 | O | 16 | Instr[15:0]。 |
| j | O | 26 | Instr[25:0]。 |

3.GRF

介绍

通用寄存器组，也称为寄存器文件、寄存器堆，可以存取 32 位数据，具有异步复位功能。寄存器标号为 0 到 31，其中 0 号寄存器读取的结果恒为0。

端口定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|----------|-------|----|--------------------------------|
| Rr1 | I | 5 | 指定 32 个寄存器中的一个，将其存储的数据读出到 Rd1。 |
| Rr2 | I | 5 | 指定 32 个寄存器中的一个，将其存储的数据读出到 Rd2。 |
| Wr | I | 5 | 指定 32 个寄存器中的一个，作为写入的目标寄存器。 |
| Wd | I | 32 | 写入寄存器的数据信号。 |
| RegWrite | I | 1 | 写使能信号，高电平有效。 |
| Clk | I | 1 | 时钟信号。 |
| Reset | I | 1 | 异步复位信号。 |
| Rd1 | O | 32 | 输出 Rr1 指定的寄存器中的数据。 |
| Rd2 | O | 32 | 输出 Rr2 指定的寄存器中的数据。 |

功能定义

| 序号 | 功能名称 | 功能描述 |
|----|------|---|
| 1 | 异步复位 | 当异步复位信号有效时，将所有寄存器的值设置为 0x00000000。 |
| 2 | 读数据 | 读出 Rr1 和 Rr2 地址对应寄存器中存储的数据到 Rd1 和 Rd2。 |
| 3 | 写数据 | 当 RegWrite 有效且时钟上升沿到来时，将 Wd 的数据写入 Wr 对应的寄存器中。 |

4.ALU

介绍

算术逻辑单元，提供 32 位按位与、按位或、加法、减法、判断相等的功能。

端口定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|--------|-------|----|----------------------|
| A | I | 32 | 参与 ALU 计算的第一个值。 |
| B | I | 32 | 参与 ALU 计算的第二个值。 |
| ALUOp | I | 4 | ALU 功能的选择信号，具体见功能定义。 |
| Result | O | 32 | ALU 的计算结果。 |
| Zero | O | 1 | 当 A = B 时为 1，否则为 0。 |

功能定义

| 序号 | 功能名称 | 功能描述 |
|----|------|----------------------------------|
| 1 | 按位与 | 当 ALUOp = 0000 时，Result = A & B。 |
| 2 | 按位或 | 当 ALUOp = 0001 时，Result = A B。 |
| 3 | 加法 | 当 ALUOp = 0010 时，Result = A + B。 |
| 4 | 减法 | 当 ALUOp = 0011 时，Result = A - B。 |
| 5 | 判断相等 | 当 A = B 时，Zero 为 1，否则为 0。 |

5.DM

介绍

数据存储器，可以存取 32 位数据，容量为 32bit * 32，具有异步复位功能，起始地址为 0x00000000。

端口定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|-------|-------|----|------------------|
| Addr | I | 5 | 读取或写入数据的地址。 |
| Write | I | 32 | 写入 DM 中的数据。 |
| store | I | 1 | 写入数据信号，高电平有效。 |
| clock | I | 1 | 时钟信号。 |
| load | I | 1 | 读取数据信号，高电平有效。 |
| clear | I | 1 | 异步复位信号。 |
| Read | O | 32 | 输出 Addr 指定地址的数据。 |

功能定义

| 序号 | 功能名称 | 功能描述 |
|----|------|--|
| 1 | 异步复位 | 当异步复位信号有效时，将 DM 中所有值设为 0x00000000。 |
| 2 | 读数据 | 当 load 有效时，读出 Addr 地址对应的数据到 Read。 |
| 3 | 写数据 | 当 store有效且时钟上升沿到来时，将 Write 的数据写入 Addr 对应的地址。 |

6.EXT

介绍

位数扩展器，可以将 16 位数据做无符号扩展和符号扩展到 32 位。

端口定义

| 端口 | 输入/输出 | 位宽 | 描述 |
|-----|-------|----|---------------------|
| i16 | I | 16 | 需要扩展的 16 位数据。 |
| u32 | O | 32 | 将输入做无符号扩展到 32 位的结果。 |
| s32 | O | 32 | 将输入做符号扩展到 32 位的结果。 |

功能定义

| 序号 | 功能名称 | 功能描述 |
|----|-------|----------------------------------|
| 1 | 无符号扩展 | 将 i16 输入的 16 位数据做无符号扩展，由 u32 输出。 |
| 2 | 符号扩展 | 将 i16 输入的 16 位数据无符号扩展，由 s32 输出。 |

7.Controller

介绍

控制器，产生控制信号，内部使用与或门阵列构造。

端口与功能定义

| 端口 | 输入/ 输出 | 位 宽 | 描述 |
|----------|-----------|--------|--|
| Op | I | 6 | 所有指令的操作码，对应 Instr[31:26]。 |
| Func | I | 6 | R 指令中辅助识别的操作码，对应 Instr[5:0]。 |
| Branch | O | 1 | 0: PC = PC + 4。 1: 如果 Zero = 1, PC 跳转到 beq 指令对应的跳转地址；否则 依旧执行 PC = PC + 4。 |
| MemtoReg | O | 2 | 00: GRF 的 Wd 输入端选择 ALU 计算结果。 01: GRF 的 Wd 输入端选择 DM 数据输出端。 10: GRF 的 Wd 输入端选择将 16 位立即数加载到高位的结果。 |
| MemWrite | O | 1 | 0: DM 的写使能端无效。 1: DM 的写使能端有效。 |
| MemRead | O | 1 | 0: DM 的读使能端无效。 1: DM 的读使能端有效。 |
| ALUOp | O | 4 | ALU 功能的选择信号，具体见 ALU 模块的功能定义。 |
| ALUSrc | O | 2 | 00: ALU 的 B 输入端选择 GRF 的 Rd2 输出端。 01: ALU 的 B 输入端选择 EXT 的 u32 输出端。 10: ALU 的 B 输入端选择 EXT 的 s32 输出端。 |
| RegWrite | O | 1 | 0: GRF 的写使能端无效。 1: GRF 的写使能端有效。 |
| RegDst | O | 1 | 0: GRF 的 Wr 输入端选择 InstrSplitter 的 Rt 输出端。 1: GRF 的 Wr 输入端选择 InstrSplitter 的 Rd 输出端。 |

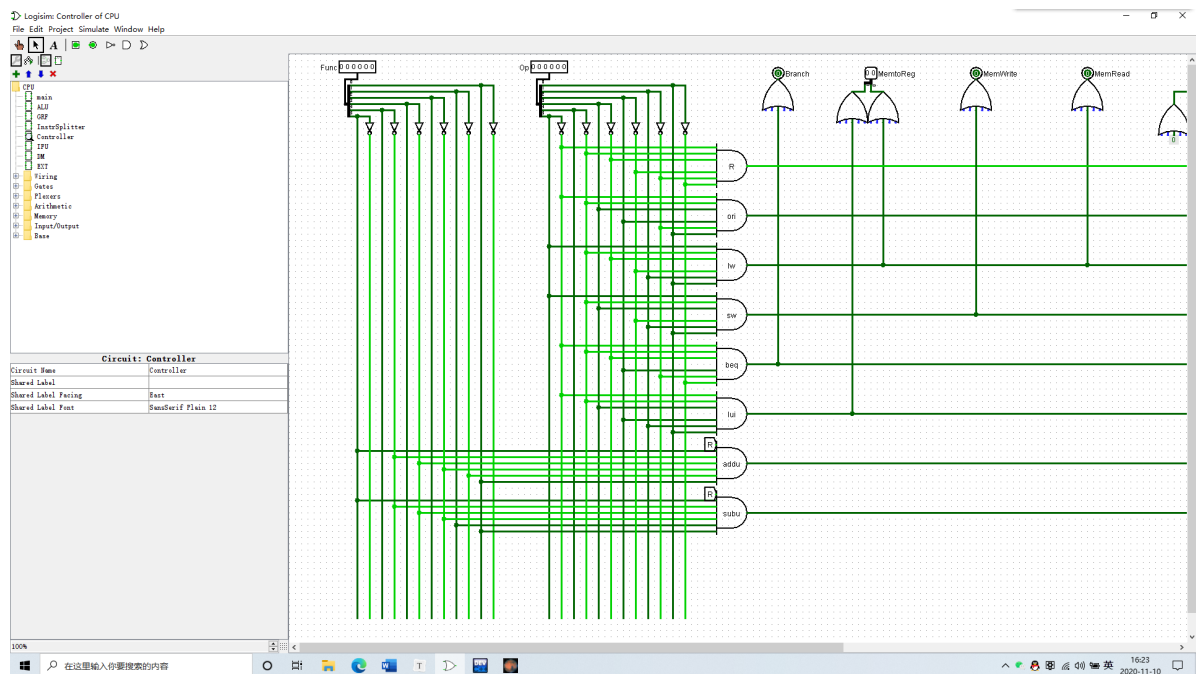
控制信号真值表

| | | R | addu | subu | ori | lw | sw | beq | lui |
|-------------|------|--------|--------|--------|--------|--------|--------|--------|--------|
| | Op | 000000 | | | 001101 | 100011 | 101011 | 000100 | 001111 |
| | Func | | 100001 | 100011 | | | | | |
| | | | | | | | | | |
| Branch | | 0 | | | 0 | 0 | 0 | 1 | 0 |
| MemtoReg[1] | | 0 | | | 0 | 0 | 0 | 0 | 1 |
| MemtoReg[0] | | 0 | | | 0 | 1 | 0 | 0 | 0 |
| MemWrite | | 0 | | | 0 | 0 | 1 | 0 | 0 |
| MemRead | | 0 | | | 0 | 1 | 0 | 0 | 0 |
| ALUOp[3] | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ALUOp[2] | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ALUOp[1] | | | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| ALUOp[0] | | | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| ALUSrc[1] | | 0 | | | 0 | 1 | 1 | 0 | 0 |
| ALUSrc[0] | | 0 | | | 1 | 0 | 0 | 0 | 1 |
| RegWrite | | 1 | | | 1 | 1 | 0 | 0 | 1 |
| RegDst | | 1 | | | 0 | 0 | 0 | 0 | 0 |

重要机制实现方法

使用与或门阵列构造控制信号

首先根据所有指令的 Op 和 Func 使用与门确定是哪一条指令，如下图所示：



再列出上面的控制信号真值表，使用或门将每一位信号与需要它的指令相连，如下图所示：


```
ori $31, $31, 31
```

2. 测试 sw 指令

```
ori $1, $1, 1
sw $1, 4($0)
sw $1, 8($0)
sw $1, 12($0)
sw $1, 16($0)
sw $1, 20($0)
sw $1, 24($0)
sw $1, 28($0)
sw $1, 32($0)
sw $1, 36($0)
sw $1, 40($0)
sw $1, 44($0)
sw $1, 48($0)
sw $1, 52($0)
sw $1, 56($0)
sw $1, 60($0)
sw $1, 64($0)
sw $1, 68($0)
sw $1, 72($0)
sw $1, 76($0)
sw $1, 80($0)
sw $1, 84($0)
sw $1, 88($0)
sw $1, 92($0)
sw $1, 96($0)
sw $1, 100($0)
sw $1, 104($0)
sw $1, 108($0)
sw $1, 112($0)
sw $1, 116($0)
sw $1, 120($0)
sw $1, 124($0)
```

3. 测试 lw 指令

```
ori $1, $1, 1
sw $1, 0($0)
lw $2, 0($0)
lw $3, 0($0)
lw $4, 0($0)
lw $5, 0($0)
lw $6, 0($0)
lw $7, 0($0)
lw $8, 0($0)
lw $9, 0($0)
lw $10, 0($0)
lw $11, 0($0)
lw $12, 0($0)
lw $13, 0($0)
lw $14, 0($0)
lw $15, 0($0)
lw $16, 0($0)
```



```
lw $17, 0($0)
lw $18, 0($0)
lw $19, 0($0)
lw $20, 0($0)
lw $21, 0($0)
lw $22, 0($0)
lw $23, 0($0)
lw $24, 0($0)
lw $25, 0($0)
lw $26, 0($0)
lw $27, 0($0)
lw $28, 0($0)
lw $29, 0($0)
lw $30, 0($0)
lw $31, 0($0)
```

4.测试 addu, subu, beq, lui, nop 指令

```
start:
ori $1, $1, 1
addu $1, $1, $1
addu $2, $1, $1
subu $3, $2, $1
lui $4, 1023
nop
beq $1, $3, start
```

自动测试工具

测试样例生成器

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    //测试 ori 指令
    for(int i = 0;i < 32;i++) printf("ori %d, %d, %d\n", i, i, i);

    //测试 sw 指令
    printf("ori $1, $1, 1\n");
    for(int i = 1;i < 32;i++){
        printf("sw %d, %d($0)\n", 1, i * 4);
    }

    //测试 lw指令
    printf("ori $1, $1, 1\n");
    printf("sw $1, 0($0)\n");
    for(int i = 2;i < 32;i++){
        printf("lw %d, %d($0)\n", i, 0);
    }
}
```

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 在指令导入后并不需要写入操作，只需进行读入操作，且实际生产过程中 IM 需要的存储容量较大，因此可以使用 ROM；DM 既需要读入又需要写入，且实际生产过程中 DM 需要的存储容量很大，无法全部使用 Register，因此使用 RAM 合理；GRF 需要的存储容量较小，且读取需要很高的速度，因此使用Register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 空指令的 Op 和 Func 均为 000000，根据现有控制信号真值表，可以发现不会它不属于任何一个指令，从而产生的控制信号不具有实际意义，因此不会进行任何有效行为，所以不需要额外将其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。
 - 可以直接减掉机器码中地址的偏移量，例如机器码中 DM 的地址范围为 [0x30000000, 0x3FFFFFFF]，则可以将其整体减掉一个 0x30000000，映射到 [0x00000000, 0x0FFFFFFF]。
 - 当然，因为本 CPU 需要支持的所有跳转指令均为**相对**的，因此即使 PC 的起始地址不为 0 也不受影响。所以我们可以直接在 MARS 中将 DM 的起始地址设为 0，这样就可以免去手工修改的麻烦。
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。
 - 优点：形式验证是对指定描述的所有可能的情况进行数学上的验证，而不是仅仅对其中的部分情况进行验证，而仅仅通过测试无法枚举完所有的情况（例如所有不超过 32 条的指令的 MIPS 汇编程序），因此形式验证更加完备和严谨。
 - 缺点：形式验证因为需要借助数学的方法，相比直接测试而言更加复杂和繁琐，需要耗费的精力更多。