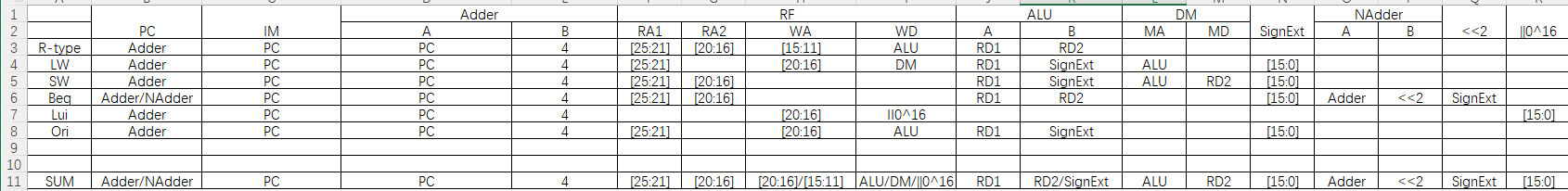
D单周期CPU

# 通路设计：

**数据通路设计表格**

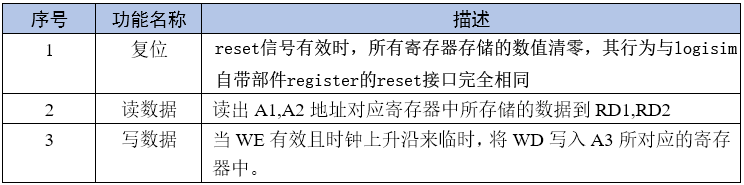


GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

GRF中包含32个32位寄存器，分别对应0~31号寄存器，其中0号寄存器读取的结果恒为0。具体模块端口定义如下：



模块功能定义如下：



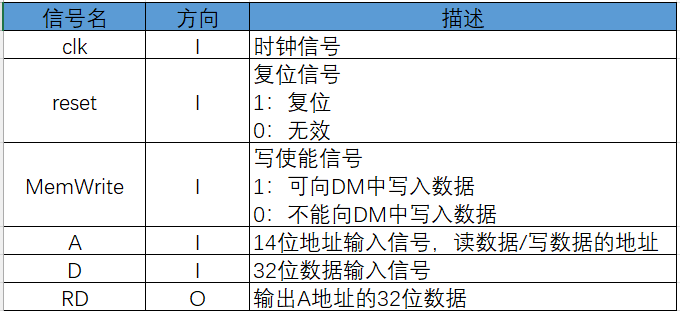
ALU（算术逻辑单元）

模块端口定义如下：



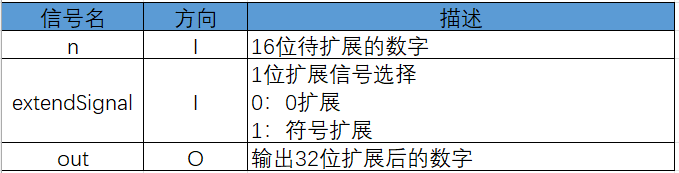
DM（数据存储器）

模块端口定义如下：



EXT（扩展单元）

模块端口定义如下：



下为原表格以便更改：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PC | IM | Adder | | RF | | | | ALU | | DM | | SignExt | NAdder | | <<2 | ||0^16 |
|  | A | B | RA1 | RA2 | WA | WD | A | B | MA | MD | A | B |
| R-type | Adder | PC | PC | 4 | [25:21] | [20:16] | [15:11] | ALU | RD1 | RD2 |  |  |  |  |  |  |  |
| LW | Adder | PC | PC | 4 | [25:21] |  | [20:16] | DM | RD1 | SignExt | ALU |  | [15:0] |  |  |  |  |
| SW | Adder | PC | PC | 4 | [25:21] | [20:16] |  |  | RD1 | SignExt | ALU | RD2 | [15:0] |  |  |  |  |
| Beq | Adder/NAdder | PC | PC | 4 | [25:21] | [20:16] |  |  | RD1 | RD2 |  |  | [15:0] | Adder | <<2 | SignExt |  |
| Lui | Adder | PC | PC | 4 |  |  | [20:16] | II0^16 |  |  |  |  |  |  |  |  | [15:0] |
| Ori | Adder | PC | PC | 4 | [25:21] |  | [20:16] | ALU | RD1 | SignExt |  |  | [15:0] |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| SUM | Adder/NAdder | PC | PC | 4 | [25:21] | [20:16] | [20:16]/[15:11] | ALU/DM/||0^16 | RD1 | RD2/SignExt | ALU | RD2 | [15:0] | Adder | <<2 | SignExt | [15:0] |

# 控制器设计：

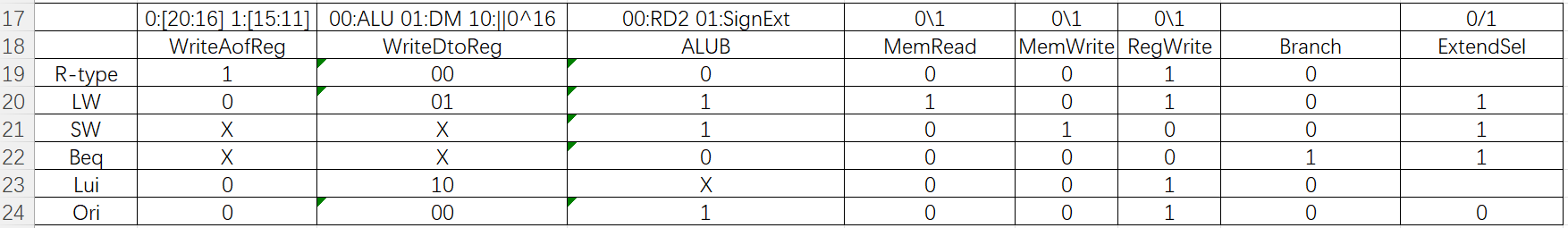
控制器分成两部分：**主控单元**和**ALU控制单元**

**主控单元：**

**输入：Op（指令31:26位）**

**输出：8个控制信号 ALUOP（ALU控制单元所需的2位输入）**

各条指令对应控制信号如下：



**控制信号**含义：

WriteAofReg：寄存器堆写入端地址选择

WriteDtoReg：寄存器堆写入端数据源选择

ALUB：ALU输入端B数据源选择

MemRead： DM读数据（输出）

MemWrite：DM写数据（输入）

RegWrite：Reg写数据

Branch：是否需要跳转分支

ExtendSel：符号拓展还是零拓展

**ALUOP** 指明ALU的运算类型：

00：访存指令所需加法

01：beq指令所需减法

10：R型指令功能码决定

11：ori指令所需或

下为原表格以便更改：

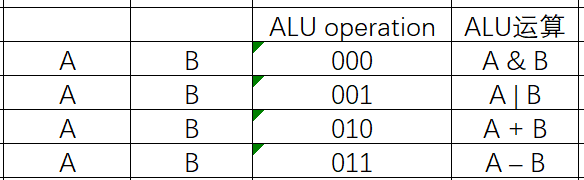
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0:[20:16] 1:[15:11] | 00:ALU 01:DM 10:||0^16 | 00:RD2 01:SignExt | 0\1 | 0\1 | 0\1 |  | 0/1 |
|  | WriteAofReg | WriteDtoReg | ALUB | MemRead | MemWrite | RegWrite | Branch | ExtendSel |
| R-type | 1 | 00 | 0 | 0 | 0 | 1 | 0 |  |
| LW | 0 | 01 | 1 | 1 | 0 | 1 | 0 | 1 |
| SW | X | X | 1 | 0 | 1 | 0 | 0 | 1 |
| Beq | X | X | 0 | 0 | 0 | 0 | 1 | 1 |
| Lui | 0 | 10 | X | 0 | 0 | 1 | 0 |  |
| Ori | 0 | 00 | 1 | 0 | 0 | 1 | 0 | 0 |

**ALU控制单元：**

**输入：ALUOP（主控单元生成） Func（功能码字段 指令5:0位)**

**输出：ALU operation（ALU运算控制信号 4位）**

各个运算对应ALU operation如下：

****

下为原表格以便更改：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | ALU operation | ALU运算 |
| A | B | 000 | A & B |
| A | B | 001 | A | B |
| A | B | 010 | A + B |
| A | B | 011 | A – B |

# 测试方案：

测试样例(在pre所给参考基础上做部分增添)：

Pre所给样例：  
ori $a0, $0, 123

ori $a1, $a0, 456

lui $a2, 123 # 符号位为 0

lui $a3, 0xffff # 符号位为 1

ori $a3, $a3, 0xffff # $a3 = -1

add $s0, $a0, $a2 # 正正

add $s1, $a0, $a3 # 正负

add $s2, $a3, $a3 # 负负

ori $t0, $0, 0x0000

sw $a0, 0($t0)

sw $a1, 4($t0)

sw $a2, 8($t0)

sw $a3, 12($t0)

sw $s0, 16($t0)

sw $s1, 20($t0)

sw $s2, 24($t0)

lw $a0, 0($t0)

lw $a1, 12($t0)

sw $a0, 28($t0)

sw $a1, 32($t0)

ori $a0, $0, 1

ori $a1, $0, 2

ori $a2, $0, 1

beq $a0, $a1, loop1 # 不相等

beq $a0, $a2, loop2 # 相等

loop1:sw $a0, 36($t0)

loop2:sw $a1, 40($t0)

添加样例：

ori $t1, 100

sw $s0, -4($t1) #offset为负数

lw $0, 0($t0) #对于$0操作

loop3:

add $t0, $t0, $a0

beq $a0, $t0, loop3 # 相等 且目标在此指令之前

# 思考题：

1. 上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

状态存储：PC GRF DM

状态转移：IM ALU EXT 主控单元 ALU控制单元

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

ROM作为只读存储器，适合用于存储固定的指令集。指令在ROM中被预先存储，CPU可以按照指令地址从ROM中读取指令。

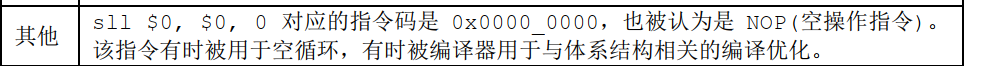
RAM能够读写数据。允许CPU在运行时动态地读写数据，适用于存储变量和临时数据。

寄存器具有非常快的读写速度，可以在CPU内部进行快速的数据操作。寄存器文件通常包含多个通用寄存器，用于存储中间结果、函数参数、局部变量等，可以减少对内存的访问次数，提高CPU的执行效率。

1. 在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

Controller（控制器）分成两部分：主控单元和ALU控制单元

1. 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？



寄存器$0的值始终为0，不会被修改

1. 阅读 Pre 的 [“MIPS 指令集及汇编语言”](http://cscore.buaa.edu.cn/tutorial/mips/mips-6/mips6-1/) 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。
   1. 存取类指令：缺少测试目标寄存器是$0的情况；offset为负数的情况
   2. 没有覆盖到所有可能的边界情况和异常情况。比如，对寄存器来说，32位数边界附近的数。
   3. Beq指令：跳转，且目标在此跳转指令之前