设计草稿

将32位单周期CPU分成IFU，GRF，ALU，DM，EXT，Controller六个部分，分别对各个部分进行设计

1. IFU部分包括PC和IM（Introduction Memory），IM地址位宽为5

端口	方向	位宽	功能
GO	input	1	为1时根据offset进行跳转
RESET	input	1	
op	output	[31:26]	
rs	output	[25:21]	
rt	output	[20:16]	
rd	output	[15:11]	
shamt	output	[10:6]	
funct	output	[5:0]	
imm	output	[15:0]	
addr	output	[25:0]	

2. GRF包括32个32位寄存器

端口	方向	位宽
rs	input	[4:0]
rt	input	[4:0]
rd	input	[4:0]
WD	input	[31:0]
CLK	input	1
RegWrite	input	1

端口	方向	位宽
RESET	input	1
RD1	output	[31:0]
RD2	output	[31:0]

3. ALU实现加、减、或、左移运算

端口	方向	位宽
A	input	[31:0]
B	input	[31:0]
ALUctrl	input	[3:0]
ZERO	output	1
C	output	[31:0]

4. DM（Data Memory），位宽为5

端口	方向	位宽
A	input	[4:0]
D	input	[31:0]
WE	input	1
CLK	input	1
RESET	input	1
D	output	[31:0]

5. EXT可以进行符号扩展和0扩展

端口	方向	位宽
ExtOp	input	1
imm16	input	[15:0]

端口	方向	位宽
ext32	output	[31:0]

6. Controller控制通路

端口	方向	位宽
op	input	[5:0]
funct	input	[5:0]
RegDst	output	1
ALUsrc	output	1
MemtoReg	output	1
MemWrite	output	1
Branch	output	1
ExtOp	output	1
LuiOffSet	output	1
ALUctrl	output	1
~Branch	output	1

设计不同指令下各个通路的多路选择器选择信号

测试方案

`add`指令

```
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
```

`sub`指令

```
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
sub $s0, $a0, $a2      # 正正
sub $s1, $a0, $a3      # 正负
sub $s2, $a3, $a3      # 负负
```

`lui`指令

```
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123            # 符号位为 0
lui $a3, 0xffff         # 符号位为 1
```

`sw`指令

```
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
sub $s0, $a0, $a2      # 正正
sub $s1, $a0, $a3      # 正负
sub $s2, $a3, $a3      # 负负
```

```
ori $t0, $0, 0x0008
sw $a0, -8($t0)
sw $a1, -4($t0)
sw $a2, 0($t0)
sw $a3, 12($t0)
sw $s0, 16($t0)
```

`lw` 指令

```
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
sub $s0, $a0, $a2      # 正正
sub $s1, $a0, $a3      # 正负
sub $s2, $a3, $a3      # 负负
ori $t0, $0, 0x0008
sw $a0, -8($t0)
sw $a1, -4($t0)
lw $a2, -4($t0)
lw $a3, -8($t0)
sw $a0, 8($t0)
sw $a1, 0($t0)
lw $a2, 0($t0)
lw $a3, 8($t0)
```

`ori`指令

```
ori $a0, $0, 123
ori $a1, $a0, 456
```

`beq` 指令

```
a:
beq $t0, $t0, c
b:
beq $t0, $t0, b
c:
beq $t0, $t0, b
```

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，请大家画出单周期 CPU 对应有限状态机的状态转移图，并谈谈它和我们之前见过的状态转移图有什么不同。

(图用markdown画不好~)

以splitter为界，“上下游”分别是一个Mealy型有限状态机。

上游：若无跳转指令（b类j类），PC = PC + 4；否则跳转到目标位置。跳转指令可视为输入信号，当前PC值可视为当前状态，PC改变后输出的instruction视为输出。满足Mealy型有限状态机。

下游：instruction分解的各个信号，通过controller控制各个通路的选择和选择不同的寄存器，然后经过ALU的逻辑运算，得到结果并在下一个时钟沿写入相应的寄存器或存储器中。整个下游部分可视作一个mealy型状态机，instruction是输入，GRF和DM的组合是状态，跳转指令可视为输出。

2. 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。

在本题中合理，ALU对DM进行读写，CPU运行过程中不对IM进行写入，GRF高效的需要读写。

但 实际上，我们通过I/O接口进行读写指令，所以IM也应该使用RAM。

3. 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。

   无

4. 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？

   寄存器写入信号RegWrite、数据存储器读写信号MemWrite、MemToReg均为0，接受nop信号时即使ALU在进行相关的计算，但并不会改变GRF和DM中的数值。

5. 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。

   当地址大于0x3000时减去0x3000

6. 阅读 Pre 的 “MIPS 指令集及汇编语言” 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. sw lw没有负数偏移量进行测试
   2. beq没有对后跳进行测试
   3. 没有对sub进行测试！！！！！！！！！！