CPU设计文档

CPU流程，草稿

1. 取指令（PC,IM)
2. 指令译码(CU)
3. 指令执行(GRF,ALU)
4. 储存器访问(DM)
5. 结果写回(GRF)
6. 循环1-5

支持指令

add, sub, ori, lw, sw, beq, lui, j, nop

(其中add，sub为无符号加减法)

使用模块

一、IFU（Instruction Fetch Unit)

- 包括PC和IM
- IM用 ROM 实现，容量为 32bit × 32字。（5位地址）
- PC始终为4的倍数（字节寻址，32bit=4字节），所以取32位地址中2-6位对应ROM中指令地址

端口名	输入/输出	位宽	功能
clk	in	1	时钟信号
reset	in	1	异步复位
PCSrc	in	1	下一指令地址选择信号
Jump	in	1	跳转控制信号
shiftResult	in	32	偏移后的指令地址
jumpAdd	in	32	直接跳转的指令地址
D	out	32	输出读取指令

三、GRF（General Register File)

- 存储$0~$31这32个寄存器数据
- 具有异步复位，读、写功能
- 最多一次同时取出两个寄存器的值运算，存入一个寄存器的值
- $0寄存器输入为常数0

端口名	输入/输出	位宽	功能

端口名	输入/输出	位宽	功能
clk	in	1	时钟信号
reset	in	1	异步复位
WE	in	1	写入使能信号
A1	in	5	输出数据到RD1的寄存器的地址
A2	in	5	输出数据到RD2的寄存器的地址
WA	in	5	输入到寄存器的地址
WD	in	32	写入的数据
RD1	out	32	RD1输出的数据
RD2	out	32	RD2输出的数据

四、ALU（Arithmetic & logical Unit）

- 提供 32 位加、减、或运算及大小比较功能，
- 加减法按无符号处理

端口名	输入/输出	位宽	功能
op	in	3	ALU功能控制信号 000：加，001：减，010：或，比较：任意
A	in	32	输入1
B	in	32	输入2
Y	out	32	输出
Greater	out	1	A>B
Equal	out	1	A=B
Less	out	1	A<B

五、DM（Data Memory）

- 使用 RAM 实现，有**异步复位**功能，复位值为 0x00000000。容量为 32bit × 32字（5位地址），使用双端口模式
- 地址始终为4的倍数（lw/rw每次操作一个字=4字节），所以取32位地址中2-6位对应RAM地址，对于lb/rb等指令还需更改

端口名	输入/输出	位宽	功能
clk	in	1	时钟信号
reset	in	1	异步复位
WE	in	1	写入使能信号

端口名	输入/输出	位宽	功能
A	in	5	将要读/写的寄存器的地址
WD	in	32	写入的数据
RD	out	32	读出的数据

六、EXT（Bit Extender）

- 将16位偏移量/立即数拓展至32位

端口名	输入/输出	位宽	功能
EXT Select	in	1	选择符号拓展/无符号拓展（0/1）
in	in	16	16位输入
out	out	32	拓展后32位输出

七、CU（Control Unit）

- 生成所有控制信号的组合逻辑电路
- 根据每条指令的数据通路列出如下控制信号表格
- R指令控制信号为（R==0）
- 分别根据Opcode和Funct每一位和与门控制非R和R型指令的选择
- 再将控制信号用或门收集所有需要触发的指令（多连接一个常数0防止没有被选择的时候输出X）

Instuction	Opcode (in)	Funct (in)	RegWrite	RegDst	ALUsrc	Branch	MemWrite	MemToReg	EXTselect	Jump	ALUControl
add	000000	100000	1	1	0	0	0	0	0	0	000
sub	000000	100010	1	1	0	0	0	0	0	0	001
ori	001101		1	0	1	0	0	0	1	0	010
lw	100011		1	0	1	0	0	1	0	0	000
sw	101011		0	0	1	0	1	0	0	0	000
beq	000100		0	0	0	1	0	0	0	0	000
lui	001111		1	0	1	0	0	0	1	0	011
j	000010		0	0	0	0	0	0	0	1	00

测试

汇编代码

```
ori $a0,$0,1999          #ori 测试程序要实现：$0寄存器中的内容与立即数 0x000007cf进行或运算，储存在$a0寄存器中
ori $a1,$a0,111          #ori 测试程序要实现：$a0寄存器中的内容与立即数 0x0000006f进行或运算，储存在$a1寄存器中
lui $a2,12345             #lui 测试程序要实现：立即数 0x00003039 加载至 $a2 寄存器的高位
lui $a3,0xffff           #lui 测试程序要实现：立即数 0x0000ffff 加载至 $a3 寄存器的高位
ori $a3,$a3,0xffff      #ori 测试程序要实现：$a3寄存器中的内容与立即数 0x0000ffff进行或运算，储存在$a3寄存器中
```

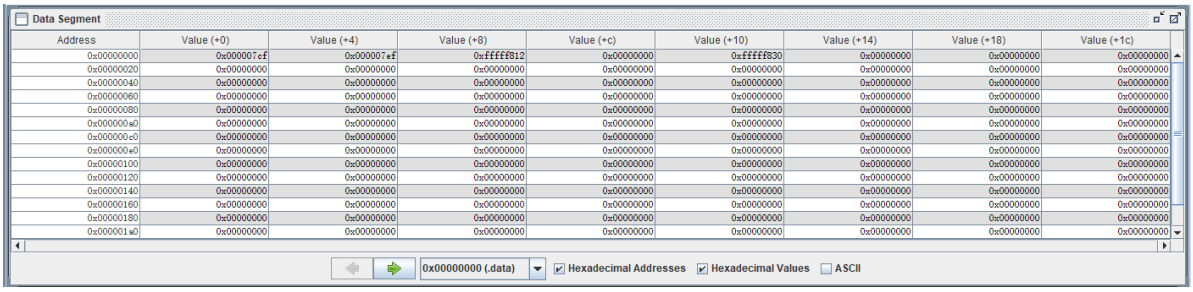
```
add $s0,$a0,$a1          #addu 测试程序要实现：a0 寄存器中的值加上a1 后存到 s0 寄存器中
add $s1,$a3,$a3          #addu 测试程序要实现：a3 寄存器中的值加上a3 后存到 s1 寄存器中
add $s2,$a3,$s0          #addu 测试程序要实现：a3 寄存器中的值加上s0 后存到 s2 寄存器中
sub $s0,$a0,$s2          #subu 测试程序要实现：a0 寄存器中的值减去 s2 寄存器中的值后存到 s0 寄存器中
sub $s1,$a3,$a3          #subu 测试程序要实现：a3 寄存器中的值减去 a3 寄存器中的值后存到 s1 寄存器中
eee:
sub $s2,$a3,$a0          #subu 测试程序要实现：a3 寄存器中的值减去 a0 寄存器中的值后存到 s2 寄存器中
sub $s3,$s2,$s1          #subu 测试程序要实现：s2 寄存器中的值减去 s1 寄存器中的值后存到 s3 寄存器中
ori $t0,$0,0x0000        #ori 测试程序要实现：$0寄存器中的内容与立即数 0x00000000进行或运算，储存在$t0寄存器中
sw $a0,0($t0)            #sw 测试程序要实现：把 a0 寄存器中值,存储到t0寄存器的值再加上偏移量 0， 所指向的 RAM 中
nop
sw $a1,4($t0)          #sw 测试程序要实现：把 a1 寄存器中值,存储到t0寄存器的值再加上偏移量 4， 所指向的 RAM 中
sw $s0,8($t0)          #sw 测试程序要实现：把 s0 寄存器中值,存储到t0寄存器的值再加上偏移量 8， 所指向的 RAM 中
sw $s1,12($t0)         #sw 测试程序要实现：把 s1 寄存器中值,存储到t0寄存器的值再加上偏移量 12， 所指向的 RAM 中
sw $s2,16($t0)         #sw 测试程序要实现：把 s2 寄存器中值,存储到t0寄存器的值再加上偏移量 16， 所指向的 RAM 中
lw $t7,0($t0)           #lw 测试程序要实现：把 t0 寄存器的值加上偏移量0 当作地址读取存储器中的值存入 t7
lw $t6,20($t0)         #lw 测试程序要实现：把 t0 寄存器的值加上偏移量20 当作地址读取存储器中的值存入 t6
ori $t0,$t0,1          #ori 测试程序要实现：$t0寄存器中的内容与立即数 0x00000001进行或运算，储存在$t0寄存器中
ori $t1,$t1,1          #ori 测试程序要实现：$t1寄存器中的内容与立即数 0x00000001进行或运算，储存在$t1寄存器中
ori $t2,$t2,2          #ori 测试程序要实现：$t2寄存器中的内容与立即数 0x00000002进行或运算，储存在$t2寄存器中
beq $t0,$t2,eee        #beq 测试程序要实现：判断 t0 的值和 t2 的值是否相等，相等转eee
beq $t0,$t1,end        #beq 测试程序要实现：判断 t0 的值和 t1 的值是否相等，相等转end
lui $t3,1111       #lui 测试程序要实现：立即数 0x00000457 加载至 $t3 寄存器的高位
end:
add $t0,$t0,$t7    #addu 测试程序要实现：t0 寄存器中的值加上t0 后存到 t0 寄存器中
j end2                 #j 测试
sw $t7,48($t0)          #sw 测试程序要实现：把 t6 寄存器中值,存储到t0寄存器的值再加上偏移量 24， 所指向的 RAM 中
lw $t5,48($t0)       #lw 测试程序要实现：把 t0 寄存器的值加上偏移量12 当作地址读取存储器中的值存入 t5
end2:
```

机器码

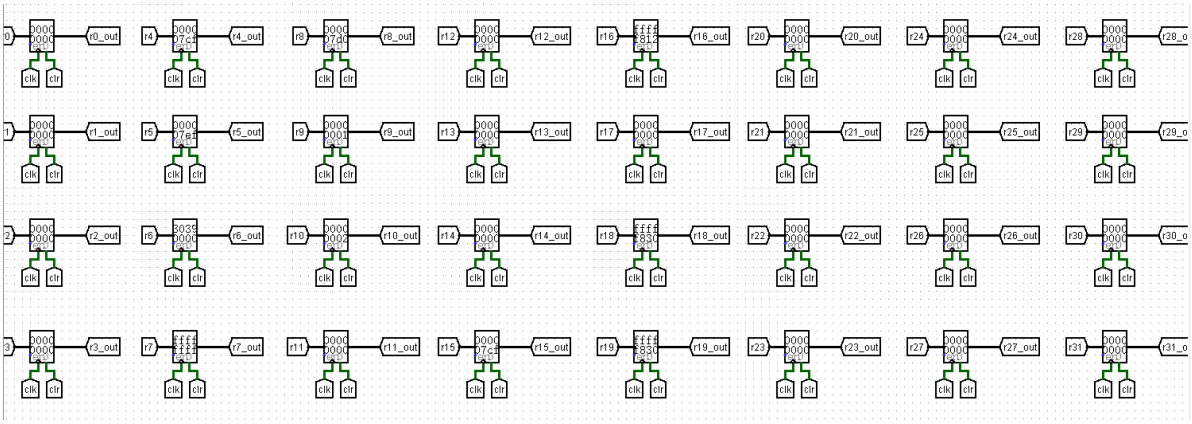
```
v2.0 raw
340407cf
3485006f
3c063039
3c07ffff
34e7ffff
00858020
00e78820
00f09020
00928022
00e78822
00e49022
02519822
34080000
ad040000
00000000
ad050004
ad100008
ad11000c
ad120010
8d0f0000
8d0e0014
35080001
35290001
354a0002
110afff1
11090001
3c0b0457
010f4020
08000c1f
ad0f0030
8d0d0030
```

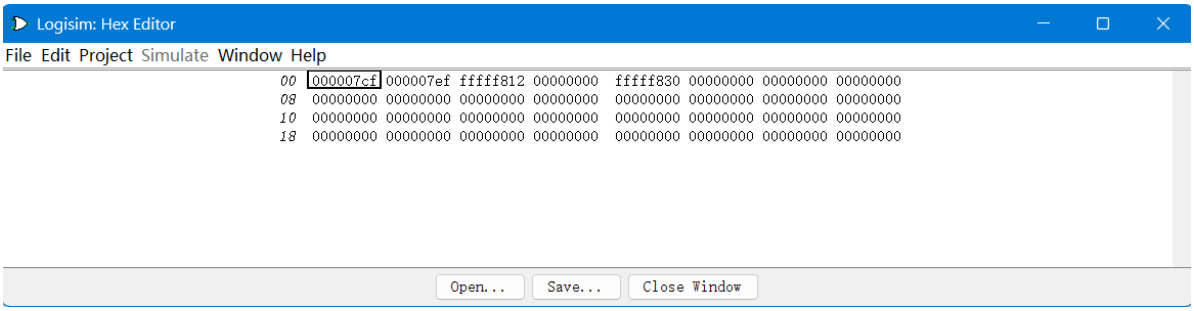
mips运行结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x000007cf
$a1	5	0x000007ef
$a2	6	0x30390000
$a3	7	0xffffffff
$t0	8	0x000007d0
$t1	9	0x00000001
$t2	10	0x00000002
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x000007cf
$s0	16	0xfffff812
$s1	17	0x00000000
$s2	18	0xfffff830
$s3	19	0xfffff830
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000307c
hi		0x00000000
lo		0x00000000



CPU运行结果





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。**
    1. 状态储存：PC, GRF, IM, DM（储存指令以及数据的状态）
    2. 状态转移：ALU, EXT, CU（组合逻辑）
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。**
    1. 合理，IM用ROM存储指令，掉电后不会丢失，并且需要人为进行修改，保证指令不会在运行中被更改；数据储存用RAM，读出和写入且访问速度快于ROM，方便每个周期读出或者写入；同时内存可能很大，不可用Register实现；GRF一共32个数据，每个周期内需要频繁同时读出和写入，用Register效率最高
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。**
    1. 暂无
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？**
    1. CU中无指令被选择时，所有控制信号输出0，不对GRF写入，不对DM写入，rs与rt均全为0，在GRF选择的均为0号寄存器，输出为0，ALU运算后仍为0，即操作为将0写回$0寄存器，所以不需要加入控制信号真值表
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。**
    1. 将地址减去0x30000000，映射到0x00000000为起始地址
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. beq指令强度不足，只包括向后跳转，没有测试向前跳转的负立即数
    2. ori指令还可测试对$0寄存器赋值，检测是否会修改
    3. 对DM和GRF，存取数据地址最好包含整个要求的地址范围和32个寄存器，保证范围设置正确，连接正确