

 [首页](#) [博客](#) [学院](#) [下载](#) [图文课](#) [论坛](#) [APP](#) [问答](#) [商城](#) [VIP会员](#) [活动](#) [招聘](#) [ITeye](#) [GitChat](#)

[写博客](#) [小程序](#) [消息](#) 

9

18

<

>

阅读数: 23705

得转载。 <https://blog.csdn.net/zhaokx3/article/details/51494271>

实现一系列的指令功能，但需要指出的是何为多周期（注意与前面写道的单周期的区别，这也是设计的关键之处）

过程分成几个阶段，每个阶段用一个时钟去完成，然后开始下一条指令的执行，而每种指令执行时所用的时钟数不尽相同，这就是所谓的多周期CPU。

台我们的多周期CPU设计之路（可以随时对应单周期的设计，注意联系与区别）。

rs(5位)	rt(5位)	rd(5位)	reserved
--------	--------	--------	----------

rs(5位)	rt(5位)	rd(5位)	reserved
--------	--------	--------	----------

rs(5位)	rt(5位)	immediate(16位)
--------	--------	----------------

mediate

rs(5位)	rt(5位)	rd(5位)	reserved
--------	--------	--------	----------

rs(5位)	rt(5位)	rd(5位)	reserved
--------	--------	--------	----------

rs(5位)	rt(5位)	immediate
--------	--------	-----------

mediate

rs(5位)	未用	rd(5位)	sa	reserved
--------	----	--------	----	----------


, 左移sa位, (zero-extend)sa


9


18









	rs(5位)	00000	rd(5位)	reserved
--	--------	-------	--------	----------

	rs(5位)	rt(5位)	rd(5位)	reserved
--	--------	--------	--------	----------

rd=0

	rs(5位)	rt(5位)	immediate(16位)
--	--------	--------	----------------

id)immediate]< - rt

	rs(5位)	rt(5位)	immediate(16位)
--	--------	--------	----------------

xtend)immediate]

明: immediate是从pc+4开始和转移到的指令之间间隔条数)

	rs(5位)	rt(5位)	immediate(16位)
--	--------	--------	----------------

gn-extend)immediate <<2

	addr[27..2]
--	-------------

,0,0), 转移

	rs(5位)	未用	未用	reserved
--	--------	----	----	----------

addr[27..2]

28],addr[27..2],0,0); \$31< - pc+4, 返回地址设置; 子程序返回, 需用指令 jr \$31。

0000000000000000000000000000(26位)

9

18

<

>

c中的指令地址，从存储器中取出一条指令，同时，pc根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送变换才送入pc。

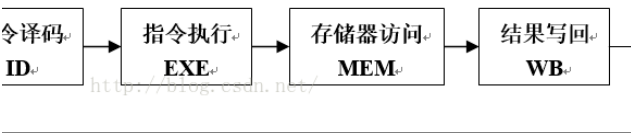
得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

果或者访问存储器中得到的数据写回相应的目的寄存器中。

，这样一条指令的执行最长需要五个(小)时钟周期才能完成，但具体情况怎样？要根据该条指令的情况而定，有些指令不需要五个时钟周期的，这就是多周期的CPU。



11 10 6 5 0

rs	rt	rd	sa	func
----	----	----	----	------

5位 6位

5 0

rs	rt	immediate
----	----	-----------

16位

0

address

址地址 (编号) 是00000~11111, 00~1F;

操作数寄存器, 寄存器地址 (同上) ;

址 (同上) ;

指令用于指定移多少位；
中（R类型）用来指定指令的功能；
符号的逻辑操作数、有符号的算术操作数、数据加载（Load）/数据保存（Store）指令的数据地址字节偏移量和分支指令中相对程序计数器（PC）的偏移量；

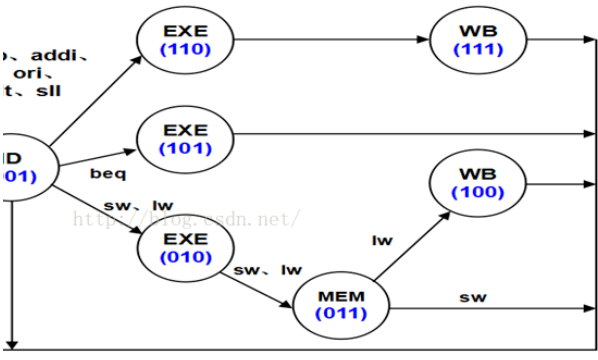


图 2 多周期 CPU 状态转移图

从状态转移到ID 和 EXE状态就是无条件的；有些是有条件的，例如ID 或 EXE状态之后不止一个状态，到底转向哪个状态由该指令功能，即指令操作码决定。每个状态代表一

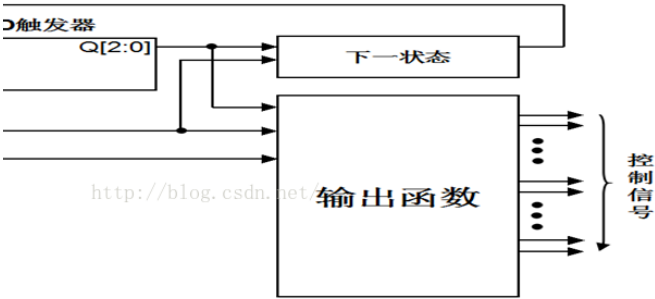


图 3 多周期 CPU 控制部件的原理结构图

3，三个D触发器用于保存当前状态，是时序逻辑电路，RST用于初始化状态“000”，另外两个部分都是组合逻辑电路，一个用于产生下一个阶段的状态，另一个用于产生下一个状态取决于指令操作码和当前状态；而每个阶段的控制信号取决于指令操作码、当前状态和反映运算结果的状态zero标志等。

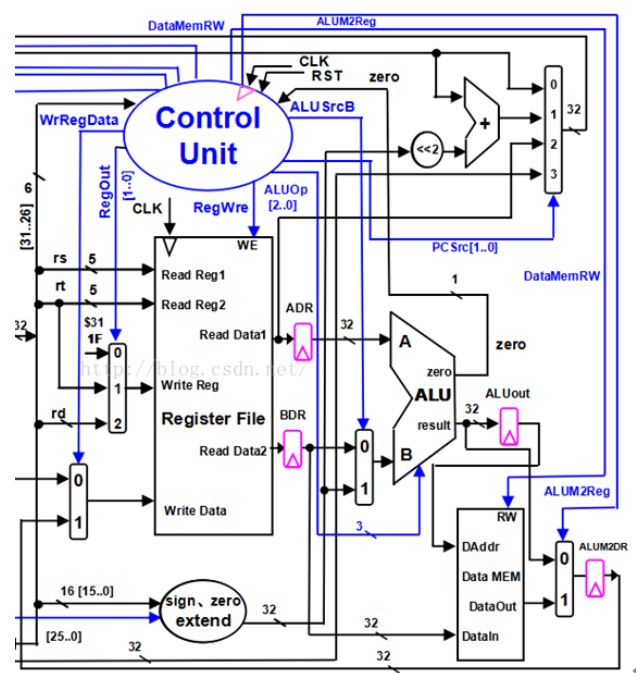


图 4 · 多周期 CPU 数据通路和控制线路图

以上完成所要求设计的指令功能的数据通路和必要的控制线路图。其中指令和数据各存储在不同存储器中，即有指令存储器和数据存储器。访问存储器时，先给出地址，然后读，也可以由时钟信号控制，但必须在图上画出来）。对于寄存器组，读操作时，给出寄存器地址（编号），输出端就直接输出相应数据；而在写操作时，在 WE 使能信号为 1 信号功能如表 1 所示，表 2 是 ALU 运算功能表。

目的是使指令代码保持稳定，还有 pc 增加写使能控制信号 pcWre，也是确保 pc 适时修改，原因都是和多周期工作的 CPU 有关。ADR、BDR、ALUout、ALUM2DR 四个寄存器数据通路，将大组合逻辑切分为若干小组合逻辑，大延时变为多个分段小延时。

表 1 控制信号作用

	状态 “1”
halt	PC 更改，相关指令：除指令 halt 外
出，相关指令：add、sub、addi、or、and、ori、	来自 sign 或 zero 扩展的立即数，相关指令：addi、ori、lw、sw、sll
出，相关指令：add、sub、addi、or、and、ori、slt、	来自数据存储器（Data MEM）的输出，相关指令：lw
相关指令：t	寄存器组寄存器写使能，相关指令：add、sub、addi、or、and、ori、move、slt、sll、lw、jal
数据来自 pc+4 (pc4)，相关指令：jal，写 \$31	写入寄存器组寄存器的数据来自存储器、寄存器组寄存器和 ALU 运算结果，相关指令：add、addi、sub、or、and、ori、slt、sll、move、lw
ta)，初始化为 0	写指令存储器
MEM)，相关指令：lw	写数据存储器，相关指令：sw
	IR 寄存器写使能。向指令存储器发出读指令代码后，这个信号也接着发出，在时钟上升沿，IR 接收从指令存储器送来的指令代码。与每条指令都相关。
(000-111)，看功能表	
关指令：add、addi、sub、or、ori、and、move、beq (zero=0)	
jn-extend) immediate，同时 zero=1，相关指令：beq	
令：jr	
,addr,0,0，相关指令：j、jal	
址，来自：	
指令：jal，用于保存返回地址 (\$31 <- pc+4)	

```
: addi, ori, lw
>: add, sub, or, and, move, slt, sll
```

l, 相关指令: sll
mediate, 相关指令: ori
mediate, 相关指令: addi, lw, sw, beq

9

18

1
, 为1写, 为0读

1
, 为1写, 为0读

端口
端口
器, 其地址输入端口 (rt、rd)
输入端口
出端口
出端口
钟上升沿写入
行的指令代码

输出1, 否则输出0

表2 ALU运算功能表

	功能	描述
	$Y = A + B$	加
	$Y = A - B$	减
	if (A<B) Y = 1; else Y = 0;	比较A与B
	$Y = A >> B$	A右移B位
	$Y = A << B$	A左移B位
	$Y = A \vee B$	或
	$Y = A \wedge B$	与

$Y = A \oplus B$	异或
------------------	----

9

18









的改进，基本框架是相同的，但是相比单周期CPU的设计最大的不同就是“多周期”，何为多周期，在实现上与单周期又有何区别？简单的来说就是每个时钟周期就执行完整指令，这就是单周期与多周期的主要区别。当然，由此也衍生出了其他的几个区别，比如，数据传输的延迟问题，增加的跳转指令；

CPU基础进行改进了。具体如下：

具体转化图见上面原理分析，例如指令add的指令状态转化是IF(000)->ID(001)->EXE(110)->WB(111)->IF；所以，需要设置两个3位的状态变量（stage）和由于指令是用来控制指令执行的，所以需要把指令状态的转变实现发在控制单元（controlUnit）中。

数据通路图中可以看出，寄存器（RegisterFile）输出处存在两个延迟（ADR）和（BDR），计算单元（ALU）的输出处存在一个延迟，数据存储器（DataMemory）输出mory）输出处存在一个延迟，当然这里延迟需要控制信号IRWre的额外控制。综上所述，前四个延迟可以设计一个叫DataLate的简单模板模块（因为它们的输入、输出完延迟可以放在INSMemory模块中。

lata,

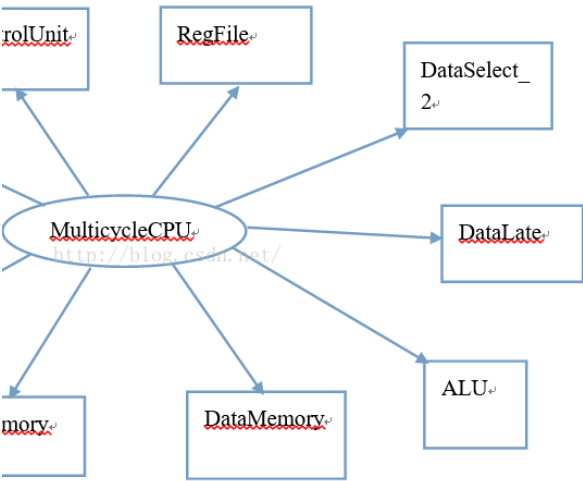
a);

根据数据通路图增添了一个如下的地址模块



本加入到相应模块中作为输入、输出。

可以画出整个多周期CPU的逻辑图。



\输出上大致相同，但具体控制内容、存在比较大的差别。

状态的关系表，如下：

 9
 18



8 of 22 5/1/19, 10:18 AM


```

[5:0] opcode,
    5 | input zero, clk, Reset,
: reg PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcB, ALUM2Reg, DataMemRW,
: reg [1:0] ExtSel, RegOut, PCSrc,
    output reg [2:0] ALUOp, state_out);

= 3'b000, // IF state
sid = 3'b001, // ID state

    exe1 = 3'b110, // add, sub, addl, or, and, ori, move, slt, sll
    exe2 = 3'b101, // beq 13 |
    exe3 = 3'b010, // sw, lw 14 |
    smem = 3'b011, // MEM state 15 |
    wb1 = 3'b111, // add, sub, addl, or, and, ori, move, slt, sll
    wb2 = 3'b100; // lw 17 |
    18 | parameter [5:0] addi = 6'b000010,

= 6'b010010,
= 6'b011000,
= 6'b000000,
= 6'b000001,
= 6'b100000,
= 6'b100111,
6'b110000,
6'b110001,
= 6'b110100,
6'b111000,
6'b111001,
6'b010000,
= 6'b010001,
= 6'b111010,
= 6'b111111;

. next_state;

= 0;
);
= 0;
0;
= 0;
= 0;
/ = 0;
2'b11;
2'b11;
?b00;
);
; if;
:= state;

clk) begin
= 0) begin
ite <= sif;
ie begin
ite <= next_state;

out = state;

` opcode) begin

ite = sid;
jin
? (opcode[5:3])
    3'b111: next_state = sif; // j, jal, jr, halt等指令
gin
rcode == 6'b110100) next_state = exe2; // beq指令
next_state = exe3; // sw, lw指令

next_state = exe1; // add, sub, slt, sll等指令

xt_state = wb1;
= sif;

```



9



18



```
= smem;
    79 |           smem: begin
(opcode == 6'b110001) next_state = wb2; // lw指令
:_state = sif; // sw指令

    sif;
    sif;
:e = sif;

begin

    && opcode != halt) PCWre = 1;

直

    IRWre = 1;

值
    || state == wb2) WrRegData = 1;
    0;

    || state == wb2 || opcode == jal) RegWre = 1;

ALUSrcB的值
fi || opcode == ori || opcode == sll || opcode == sw || opcode == lw) ALUSrcB = 1;
;

DataMemRW的值
    && opcode == sw) DataMemRW = 1;
    0;

ALUM2Reg的值
    ALUM2Reg = 1;
);

ExtSel的值
i) ExtSel = 2'b01;
== sll) ExtSel = 2'b00;
>10;

RegOut的值
.) RegOut = 2'b00;
== addi || opcode == ori || opcode == lw) RegOut = 2'b01;
>10;

PCSrc的值

'b11;
2'b11;
?'b10;

    PCSrc = 2'b01;
~c = 2'b00;

~c = 2'b00;

ALUOp的值

3'b001;
3'b101;
3'b110;
3'b101;
3'b010;
```

9

18









```

3'b100;152|          beq: ALUOp = 3'b001;
)p = 3'b000;

```

在IF阶段写数据

```

| begin

```

```

);

```



9



18



号作为输入，将结果输出，具体设计如下：

```

eadData1, ReadData2, inExt,

```

```

input [2:0] ALUOp,
output wire zero,
output reg [31:0] result);

```

```

cB? inExt : ReadData2;
result? 0 : 1);

```

```

al or ReadData2 or B or ALUOp) begin

```

```

t = ReadData1 + B; // A + B
t = ReadData1 - B; // A - B
t = (ReadData1 < B ? 1 : 0); // 比较A与B
t = ReadData1 >> B; // A右移B位
t = ReadData1 << B; // A左移B位
t = ReadData1 | B; // 或
t = ReadData1 & B; // 与
t = (~ReadData1 & B) | (ReadData1 & ~B); // 异或
0;

```

中多了一个四选一的地址数据选择器，目的在于根据控制信号正确匹配pc地址，同样输出当前PC地址，具体设计如下：

```

t, PCWre,
rc,
0] imm, addr, RDout1,
0] Address);

```

```

r negedge Reset) begin // 这里和单周期不太一样，存在延迟的问题，只有当pcWre改变的时候或者Reset改变的时候再检测
egin

```

```

e) begin
2'b00) begin

```

```
        Address = Address+4;
        13 |
    end else if (PCSrc == 2'b01) begin14 |
        Address = imm*4+Address+4;15 |
    end else if (PCSrc == 2'b10) begin16 |
        end else if (PCSrc == 2'b11) begin
            Address = addr;
        end
    end

    Address = RDout1;
```

👍9

💬18

🔖

📄

<

>

ss)

的地址，模块实现如下：

```
] in_addr,
] PC0,
    output reg [31:0] addr);

addr << 2;
in
8], mid[27:0]);
```

多了一些，包括sa扩展、立即数扩展等，扩展选择由控制信号ExtSel控制，最后输出完整32位数据。

```
] in_num,
ExtSel,
    output reg [31:0] out);

tSel) begin

    {{27{0}}, in_num[10:6]}; // 扩充 sa
    {{16{0}}, in_num[15:0]}; // 扩充立即数，如 ori指令
    {{16{in_num[15]}}, in_num[15:0]}; // 符号扩充立即数，如addi、lw、sw、beq指令
    <= {{16{in_num[15]}}, in_num[15:0]}; // 默认符号扩展
```

ory)

居数据通路图可以有如下模块设计：

```
31:0] addr, Data2,
ataMemRW,
reg [31:0] DataOut);

[0:63];
```

```
initial begin
i < 64; i = i+1) memory[i] <= 0;

2 or DataMemRW) begin
n // write data
Data2[31:24];
= Data2[23:16];
= Data2[15:8];
= Data2[7:0];
oad data
= memory[addr];
= memory[addr+1];
memory[addr+2];
memory[addr+3];
```

9

18

ry)

\$.txt) 存入当前目录，然后通过读取文件的方式将指令存储到内存中，最后实现指令的读取。其中，

令代码，如下：

序	指令代码					16进制数代码	
	op (6)	rs(5)	rt(5)	rd(5)/immediate (16)			
000008	111000	00 00000000 00000000 00000010					
	111001	11111			00000		
1,\$0,8	000010	00000	00001	0000 0000 0000 1000	=	08010008	
\$0,2	010010	00000	00010	0000 0000 0000 0010	=	48020002	
3,\$1,\$2	000000	00001	00010	00011 000000000000	=	00221800	
1,\$1,\$2	000001	00001	00010	00100 000000000000	=	04222000	
3,\$3,\$2	010001	00011	00010	00101 000000000000	=	44622800	
\$1,\$2	010000	00001	00010	00110 000000000000	=	40223000	
\$11,\$1	100000	00001	00000	01011 000000000000	=	80205800	
\$1,\$2	100111	00001	00010	00111 000000000000	=	9C223800	
\$2,\$1	100111	00010	00001	01000 000000000000	=	9C414000	
\$2,2	011000	00010	00000	00010 00010 000000	=	60401080	
1,\$2,-2 转02C	110100	00001	00010	1111 1111 1111 1110	=	D022FFFE	
0(\$3)	110000	00011	01001	0000 0000 0000 0000	=	C0690000	
00000004	111010	00 00000000 00000000 00000001			=		
1,2(\$1)	110001	00001	01010	0000 0000 0000 0010	=	C42A0002	
	111111	00000	00000	000000000000000000	=	FC000000	

二进制指令文件，从而进行存取，二进制文件如下：

帮助(H)

```
) 00000010
) 00000000
) 00001000
) 00000010
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 00000000
) 10000000
) 11111110
) 00000000
) 00000001
) 00000010
) 00000000
```



9



18



```
1:0] addr,
sMemRW, IRWre, clk,
        output reg [31:0] ins);
```

```
:127];
```

```
y_store.txt", mem);
ut = 0;
```

```
MemRW) begin
in
= mem[addr];
= mem[addr+1];
mem[addr+2];
mem[addr+3];
```

```
clk) begin
ns <= ins_out;
```

tionMemory中的rs, rt, rd作为输入，输出对应寄存器的数据，从而达到取寄存器里的数据的目的，具体设计如下：

中，为了防止0号寄存器写入数据需要在writeReg的时候多加入一个判断条件，即writeReg不等于0时写入数据。

```
] rs, rt, rd,
RegWre, WrRegData,
nput [1:0] RegOut,
        input [31:0] PC4, memData,
        output reg [31:0] data1, data2);
```

```
3 |
ter [0:31];

32; i = i+1)
ister[i] = 0;

begin

emp = 5'b11111;
01: temp = rt;
10: temp = rd;
    default temp = 0;

i_data = WrRegData? memData : PC4;
    = register[rs];
gister[rt];
5 (RegWre == 1)) begin // temp != 0 确保零号寄存器不会改变
] <= i_data;
```

9

18









lect 2)

元后面的数据选择，可见数据通路图，实现如下：

```
[31:0] A, B,
Sign,
        output wire [31:0] Get);

: A;
```

.ate)

输入输出，从数据通路图中可知此模板可在四处地方有用，已分析，所以具体模板实现如下：

```
:0] i_data,
,
g [31:0] o_data);

begin
```

模块，通过连接各个子模块来达到运行CPU的目的，整个模块设计可以如下：

```
`include "DataMemory.v"
```

```
et,  
:0] state_out,  
:0] opcode,  
output wire [4:0] rs, rt, rd,  
31:26], ins[25:21], ins[20:16], ins[15:11],  
1:0] ins, ReadData1, ReadData2, pc0, result);
```

```
ins[31:26];  
25:21];  
20:16];  
15:11];
```

```
ut1, out2, result1, i_IR, extendData, LateOut1, LateOut2, DataOut;
```

```
gOut, PCSrc;  
sMemRW, WrRegData, RegWre, ALUSrcB, DataMemRW, ALUM2Reg;
```

```
, PCWre, PCSrc, extendData, j_addr, ReadData1, pc0);
```

```
ory(pc0, InsMemRW, IRWre, clk, ins);
```

```
s[25:0], pc0, j_addr);
```

```
ins[25:21], ins[20:16], ins[15:11], clk, RegWre, WrRegData, RegOut, (pc0+4), LateOut2, ReadData1, ReadData2);
```

```
dData1, clk, out1);  
dData2, clk, out2);
```

```
s[15:0], ExtSel, extendData);
```

```
t2, extendData, ALUSrcB, ALUOp, zero, result);
```

```
result, clk, result1);
```

```
emory(result1, out2, DataMemRW, DataOut);
```

```
aselect_2(result, DataOut, ALUM2Reg, LateOut1);
```

```
(LateOut1, clk, LateOut2);
```

```
ins[31:26], zero, clk, reset,PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcB, ALUM2Reg, DataMemRW, ExtSel, RegOut, PCSrc, ALUOp, state_out)
```

入只有时钟信号clk和重置信号Reset，所以测试程序代码比较简单。（参照单周期CPU）



reset to finish

👍

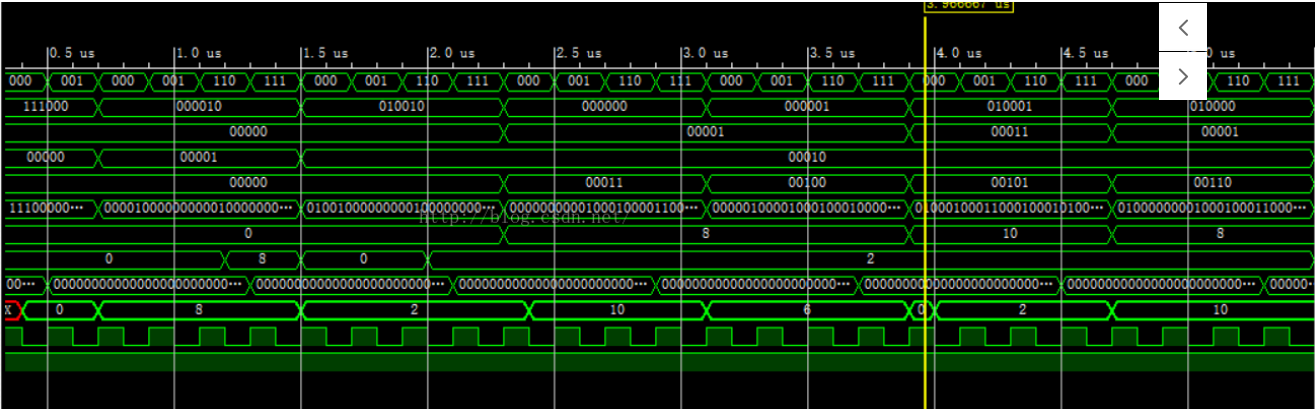
9

💬

18

🔖

📄



🔖 收藏

🔗 分享

- 件 代码部分怎么没有体现呢 (1周前 #8楼) 👍
- 母? 急用 (3个月前 #7楼) 👍
- 开始就是高阻态, 是为什么呢 (1年前 #6楼) 查看回复(1) 👍

查看 18 条热评

📅: 2017/5/29) 阅读数 9330
📄: 单周期CPU设计(Verilog)一、实验目的(1)认识和掌握多... 博文 来自: linwh8的博客

📄:PU实验 阅读数 5982
实验实验老师给我们罗列了很细致的控制线路图~然而一些... 博文 来自: Sician_Lee的...

📄:期CPU指的是将整个CPU的执行过程分成几个阶段, 每个... 博文 来自: Zhongzi_L的...

📄:设计一个多周期CPU, 该CPU至少能实现以下指令功能操... 博文 来自: llllyt的博客

- 📄:文档) 12-07
📄:ISE, 实验文档中包含状态机示意图和线路连接图 下载
- 📄:中有实验原理、实验设计、实验代码等。 06-16
📄: 下载
- 📄:与, 不包含代码, 为了老师的正常教学 实验报告只是提供参考和思路 01-27
📄: 下载

<div>方真代码</div> <div>构建寄存器堆、ALU、CONUNIT等模块，支持12条指令：add、sub、j、bne、bnq等</div>	12-28	下载	<div><div><div></div></div><div>9</div></div>
<div>解MIPS指令集编码以及指令格式基础上，自己设计数据通路（可参考图4.1或教材），实现...</div>	12-11	下载	<div><div><div></div></div><div>18</div></div>
<div>计</div> <div>验要求设计并实现一个多周期MIPSCPU，并满足如下要求... 博文 来自: saber_jk的博客</div>	阅读数 925		
<div>设计实验代码以及实验报告（vivado）</div> <div>成原理实验多周期CPU设计实验代码以及实验报告（vivado）</div>	12-26	下载	<div><div><div></div></div><div></div></div>
<div></div> <div>都是类似的我是把所有数据选择器的模块都单独拿出来，这... 博文 来自: sysu_zjl的博客</div>	阅读数 6161		
<div></div> <div>增加IR指令寄存器，目的是使指令代码保持稳定，pc写使能... 博文 来自: 晨识草的博客</div>	阅读数 767		
<div>第三步</div> <div>峰造极 基础课程要求：数字电路、计算机组成原理、程序设... 博文 来自: MarshalRUAN</div>	阅读数 265		
<div></div> <div>期CPU的基础上完成了，将每条指令只需要一个周期，切割... 博文 来自: 第五清风的博客</div>	阅读数 865		
<div></div> <div>，如果需要看到仿真的波形图，可以跑仿真，调节相关参数即可显示出来</div>	06-29	下载	
<div>S架构的CPU。分别设计指令存储器、寄存器堆、ALU、取指令部件、数据存储器、立即数处...</div>	08-08	下载	
<div>P0，具体指令参见压缩包中的PDF文件。配有54条指令仿真测试的coe文件以及每一条指令单...</div>	12-08	下载	
<div>I}，内容详尽，代码易读</div>	12-27	下载	
<div>三) —— 主存</div> <div>SRAM和DRAM两大类，均为阵列式，由于Vivado已经提... 博文 来自: とある黄鱼の...</div>	阅读数 367		
<div>signment to a non-net an is not permitted ["...</div> <div>oanon-netanisnotpermitted["&quot;C:/Users/chenxy... 博文 来自: cxy_hust的博客</div>	阅读数 1852		
<div></div> <div>所有的工作，既从指令取出，到得到结果，全部在一个时钟... 博文 来自: u014670574...</div>	阅读数 3098		
<div></div> <div>，下面我们说一下多周期。单周期每一条指令都是一个时钟... 博文 来自: stanary的博客</div>	阅读数 280		
<div></div> <div>关设计，最近总算做出一个马马虎虎的。先来说说思路。... 博文 来自: WSQPoison...</div>	阅读数 2224		
<div></div> <div>考别人的博客自己做了一遍单周期cpu后，觉得不是很难，... 博文 来自: MyCodecOd...</div>	阅读数 1436		
<div>编译器实现</div> <div>前器的简单实现代码和可执行文件，其中MIPS多周期项目使用ISE（Verilog语言），汇编译器实...</div>	05-13	下载	
<div></div> <div>就是多周期CPU了。因为有很多单周期的代码可以复用，所... 博文 来自: GoldenSpace</div>	阅读数 330		
<div></div> <div>完成，其实只想写写多周期的，无奈单周期补上才好，哈哈... 博文 来自: keep studying</div>	阅读数 3万+		

阅读数 9629
图二 主要收获：1.阻塞赋值与非阻塞赋值；2.代码测试；3.组... 博文 来自： u012373020...

y * directly; Cannot assign memory * directly 阅读数 3503
开了一个数组data_in[7:0]，导致赋值的时候出现了错误。... 博文 来自： xiao_du的博客

区别 阅读数 4907
中，wire永远是wire，就是相当于一条连线，用来连接电路... 博文 来自： 坚持

线性cpu 02-10
下载

阅读数 570
水灯的变换形式为：00000000->00000001->00000011-... 博文 来自： allen_tony的...

一代一代的传承 05-28
下载

以及测试 03-22
下载

I (10) 阅读数 239
可以分解读/修改以及读/修改/写指令来改进成对性。例如：;... 博文 来自： wuhui_gdnt...

指令 05-08
CPU 下载

i) 阅读数 703
www.cnblogs.com/shengansong/archive/2012/05/17/... 博文 来自： 爱哭的小飞熊

阅读数 2126
万，从2010年到2017年进行交易，最大仓位257.67万，净... 博文 来自： 云金记

阅读数 2万+
普林斯顿体系结构，是一种将程序指令存储器和数据存储器... 博文 来自： skywalker_le...

与Verilog语言实现 多周期CPU 09-13
源代码及多周期CPU结构图，与大家分享下。 下载

阅读数 685
Z，以下实验仅为16年报告代码下载一.实验目的(1)认识和掌... 博文 来自： Loeng

与VHDL实现 05-22
水线的设计详细流程。 下载

立CPU设计与VHDL实现) 10-22
DL实现) 下载

阅读数 1万+
Z，没有深入的进行调试，我准备在放假的时候重构一下代码... 博文 来自： linwh8的博客

l) ---logisim部件设计 阅读数 9406
的CPU的代码书写的过程中必然时刻伴随着设计图纸的需求... 博文 来自： TONNE_YAN...

01-08
单元宽度一律使用8位，即一个字节的存储单位。不能使用32位作为存储器存储单元宽度。控... 下载

六) —— 流水线 阅读数 1265
单周期CPU问题早期的RISCCPU采用了简单的单周期执行模... 博文 来自： とある黄鱼の...

单周期CPU 可以实现16条指令 11-18
以实现16条指令 下载

 9

 18









04-23
下载

9

18









Ⅱ.			
的安装与应用	博文	来自: roguesir的博客	阅读数 6万+
应用			
3/1999/09/expat/index.html 因为需要用,所以才翻译了...	博文	来自: ymj7150697...	阅读数 4万+
删工具BlockDetectUtil (仅一个类)	博文	来自: u012874222...	阅读数 3089
从着用户的体验性,而体验性好的产品自然而然会受到更多用...			
je模块(ios android icon图标自动生成处理)	博文	来自: 专注于cocos+...	阅读数 7万+
w.pythonware.com/products/pil/ 2.解压后,进入到目...			
装中文语言包	博文	来自: SweetTool的...	阅读数 6万+
5.不过,但是ubuntu的server版本也相当出色。作为习惯使...			
前做的编解码的项目总结一下。话说一年多没碰,之前做的笔...	博文	来自: 不积跬步,无...	阅读数 8137
框架	博文	来自: qilixuening的...	阅读数 1万+
现用CPU计算的Keras框架性能明显不够用了,但当时随便...			
统入门实例	博文	来自: 般若	阅读数 2万+
i, 异步加载页面,多Tab页展示,使用JSON文件模拟从后...			
nginx	博文	来自: maoyuanmin...	阅读数 4万+
一个高性能的HTTP和反向代理服务器,也是一个IMAP/PO...			
加载布局)	博文	来自: 喻志强的博客	阅读数 1万+
的问题,有需要的同学可以看一下,底部有demo下载。直...			
↑	博文	来自: 九野的博客	阅读数 59万+
去一次,两两可达) 孤立的一个点也是一个连通分量 使用ta...			
不错 http://karan.iteye.com/blog/212975	博文	来自: slwsf的专栏	阅读数 64
的问题 (与文件包位置有关)	博文	来自: 开发随笔	阅读数 19万+
大致结构如下: 核心Spring框架一个module spring-boot...			
的堆,是计算机科学中一类特殊的数据结构的统称。在队列...	博文	来自: weixin_3954...	阅读数 900
多个倒计时(最新的)	博文	来自: Websites	阅读数 46万+
时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本...			
篇文章将重点分析SNMP报文,并对不同版本 (SNMPv1、v...	博文	来自: 假装在纽约	阅读数 12万+
“%{” 和 “%}” 。例如 %{。。。 %} 即可。经典方法是用 ...	博文	来自: 知识小屋	阅读数 2万+
4188315/article/details/51734514 SQLite分页查询有三...	博文	来自: duxingzhe20...	阅读数 2443
针编号, 编号最大限定为100000。求任意两编号之间的最...	博文	来自: NYS001的专栏	阅读数 1万+
专家编程》	博文	来自: Stay Hungry,...	阅读数 1万+
是一个数学概念,如果把它运用于程序中,可以发挥很大的作...			

已

阅读数 4万+

博文 来自: lubiaopan的...

计学稳健估计opencv函数 计算机导论培训 机器学习教程
react extjs glyph 图标 区块链学习周期 java培训周期

9

18











OneDay-X

关注

原创37

粉丝65

喜欢81

评论104

等级: 博客

访问: 25万+

积分: 2138

排名: 2万+

勋章:

最新文章

2的幂次方表示

c++实现大数运算

git 使用

使用docker容器定制镜像 (image) 并部署简单 web应用

远程访问Tensorboard

个人分类

计算机组成与设计2篇

Unity3D学习9篇

c#3篇

Java6篇

操作系统 (OS)2篇

展开

归档

2018年3月2篇

2017年7月1篇

2017年5月1篇

2017年4月3篇

2017年3月7篇

展开

热门文章

单周期CPU设计
阅读数 31082

查看并修改签名证书keystore的密码， alias别名等相关参数
阅读数 29365

使用docker容器定制镜像（image）并部署简单 web应用
阅读数 23841

多周期CPU设计
阅读数 23694

生成签名证书keystore
阅读数 21030

最新评论

多周期CPU设计
cui_bao_jing: 数据通路那个图有几个部件 代码部分怎么没有体现呢

java Socket实现多人群聊...
qq_34316044: [reply]qq_44034728[/reply] 应该是断开socket连接就行吧

c++实现大数运算
weixin_41678774: sorry 没看到顺序。。

c++实现大数运算
weixin_41678774: 乘法结果不对啊

java Socket实现多人群聊...
lzjyyy: [reply]qq_44034728[/reply] 强制” ...



程序人生



CSDN资讯

QQ客服

客服论坛

kefu@csdn.net

400-660-0108

工作时间 8:30-22:00

关于我们 招聘 广告服务 网站地图

百度提供站内搜索 京ICP备19004658号

©1999-2019 北京创新乐知网络技术有限公司

网络110报警服务 经营性网站备案信息

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护

9

18

<

>

22 of 22

5/1/19, 10:18 AM