CPU 设计报告

计算机组成原理大作业

姓名：李太吉

班级：14011706

学号：2017303134

学院：软件学院

西北工业大学软件学院

2019 年 6 月 4 日

目录

§1 指令分析

1.1 指令集

我们要实现的 CPU 所应支持的指令集为以下 8 条指令。

```
mov dest,sour;[sour]— > [dest]
add dest,sour;[dest]+ = [sour]
sub dest,sour;[dest]— = [sour]
and dest,sour;[dest]& = [sour]
or dest,sour;[dest]|= [sour]
not dest;[dest] = [dest]
jmp tar;Jump to tar to run
hlt; halt but not shutdown computer
```

1.2 指令集要求

指令集的要求和 CPU 的寻址特点为:

1. 支持 0 操作数、单操作数和双操作数三种指令
2. 所有指令的两个操作数不能同时为内存操作数
3. 支持立即寻址、直接寻址、寄存器直接寻址和相对寻址四种寻址方式
4. 采用 1 字节或者 2 字节变长指令字，操作码采用定长格式
5. CPU 字长为 8 位，8 个程序员可见的寄存器，分别命名为 $r_0, \cdots, r_7$
6. 地址总线、数据总线各为 8 位，可访问 2^8 字节的地址空间

1.3 指令格式分析

首先，我们分析这 8 条指令。我们用 reg 代表寄存器，mem 代表存储器，A 代表立即数。考察 8 条指令的具体格式。

1.3.1 mov 指令

```
mov reg,reg;寄存器-> 寄存器，寄存器直接寻址
mov reg,mem;存储器-> 寄存器，寄存器直接寻址和直接寻址
mov mem,reg;寄存器-> 存储器，寄存器直接寻址和直接寻址
```

1.3.2 add 指令

```
add reg,reg;寄存器-> 寄存器，寄存器直接寻址
add reg,mem;存储器-> 寄存器，寄存器直接寻址和直接寻址
```

1.3.3 sub 指令

```
sub reg,reg;寄存器-> 寄存器，寄存器直接寻址
sub reg,mem;存储器-> 寄存器，寄存器直接寻址和直接寻址
```

1.3.4 and 指令

```
and reg,reg;寄存器-> 寄存器，寄存器直接寻址
and reg,mem;存储器-> 寄存器，寄存器直接寻址和直接寻址
```

1.3.5 or 指令

```
or reg,reg;寄存器-> 寄存器，寄存器直接寻址
or reg,mem;存储器-> 寄存器，寄存器直接寻址和直接寻址
```

1.3.6 not 指令

```
not reg;寄存器，寄存器直接寻址
```

1.3.7 jmp 指令

```
jmp A;立即数，相对寻址
```

§2 指令格式设计

指令集共有 8 条指令，操作码定长，所以我们规定操作码（OP）占 3 位，寻址模式有 4 种，则用 2 位寻址特征来代表，CPU 地址空间为 2^8 字节，则每个地址码占 8 位，寄存器共有 8 个，故寄存器编号为 3 位。

我们规定操作码和寻址特征如下：

2.1 操作码

指令	操作码
mov	000
add	001
sub	010
add	011
or	100
not	101
jmp	110
hlt	111

表 2.1: 各指令的操作码

2.2 寻址特征

寻址方式	寻址特征
reg-reg	00
reg-mem	01
mem-reg	10
reg	11

表 2.2: 寻址特征

综合以上分析，我们设计这八条指令具体格式如下。

2.3 指令格式

2.3.1 mov 指令

mov 指令一共有四种。

1. reg-reg

OP	M	R1	R2	ADD
000(3bit)	00(2bit)	3bit	3bit	5bit

表 2.3: mov: reg-reg

2. reg-mem

OP	M	R	MEM
000(3bit)	01(2bit)	3bit	8bit

表 2.4: mov: reg-mem

3. mem-reg

OP	M	R	MEM
000(3bit)	10(2bit)	3bit	8bit

表 2.5: mov: mem-reg

4. reg-立即数

OP	M	R	立即数
000(3bit)	11(2bit)	3bit	8bit

表 2.6: mov: reg-立即数

注：ADD 字段为指令字长的补齐字段，无意义。

2.3.2 add 指令

add 指令有三种。

1. reg-reg

OP	M	R1	R2	ADD
001(3bit)	00(2bit)	3bit	3bit	5bit

表 2.7: add: reg-reg

2. reg-mem

OP	M	R	MEM
001(3bit)	01(2bit)	3bit	8bit

表 2.8: add: reg-mem

3. reg-立即数

OP	M	R	立即数
001(3bit)	10(2bit)	3bit	8bit

表 2.9: add: reg-立即数

2.3.3 sub 指令

sub 指令有三种。

1. reg-reg

OP	M	R1	R2	ADD
010(3bit)	00(2bit)	3bit	3bit	5bit

表 2.10: sub: reg-reg

2. reg-mem

OP	M	R	MEM
010(3bit)	01(2bit)	3bit	8bit

表 2.11: sub: reg-mem

3. reg-立即数

OP	M	R	立即数
010(3bit)	10(2bit)	3bit	8bit

表 2.12: sub: reg-立即数

2.3.4 and 指令

and 指令有三种。

1. reg-reg

OP	M	R1	R2	ADD
100(3bit)	00(2bit)	3bit	3bit	5bit

表 2.13: and: reg-reg

2. reg-mem

OP	M	R	MEM
100(3bit)	01(2bit)	3bit	8bit

表 2.14: and: reg-mem

3. reg-立即数

OP	M	R	立即数
100(3bit)	10(2bit)	3bit	8bit

表 2.15: and: reg-立即数

2.3.5 or 指令

or 指令有三种。

1. reg-reg

OP	M	R1	R2	ADD
101(3bit)	00(2bit)	3bit	3bit	5bit

表 2.16: or: reg-reg

2. reg-mem

OP	M	R	MEM
101(3bit)	01(2bit)	3bit	8bit

表 2.17: and: reg-mem

3. reg-立即数

OP	M	R	立即数
101(3bit)	10(2bit)	3bit	8bit

表 2.18: or: reg-立即数

2.3.6 not 指令

not 指令有一种。

OP	M	R
101(3bit)	00(2bit)	3bit

表 2.19: not: reg

2.3.7 jmp 指令

jmp 指令有一种。

OP	A（立即数）
110(3bit)	5bit

表 2.20: jmp: A

2.3.8 hlt 指令

hlt 指令有一种。

OP	ADD
110(3bit)	5bit

表 2.21: hlt

§3 CPU 逻辑框图

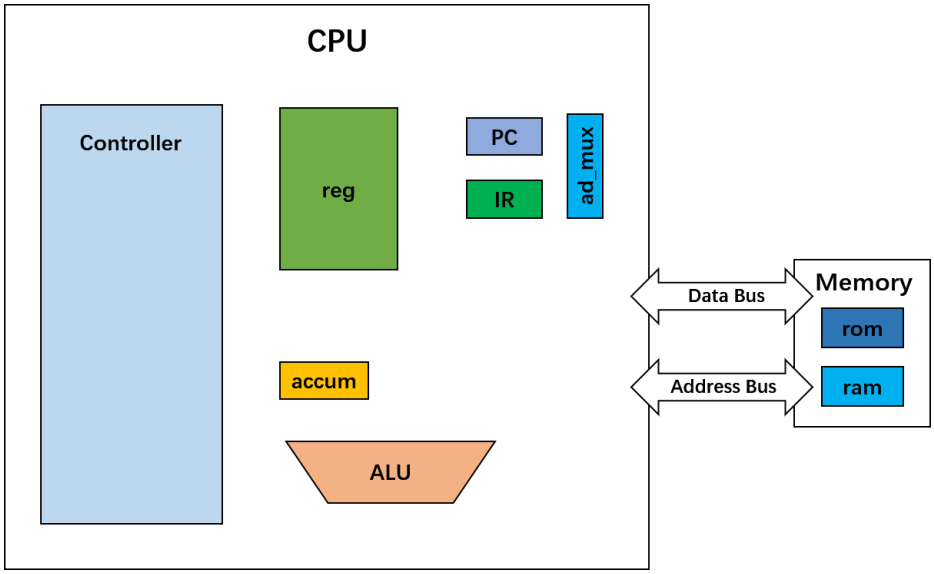

图 3.1: 组合逻辑 CPU 框图

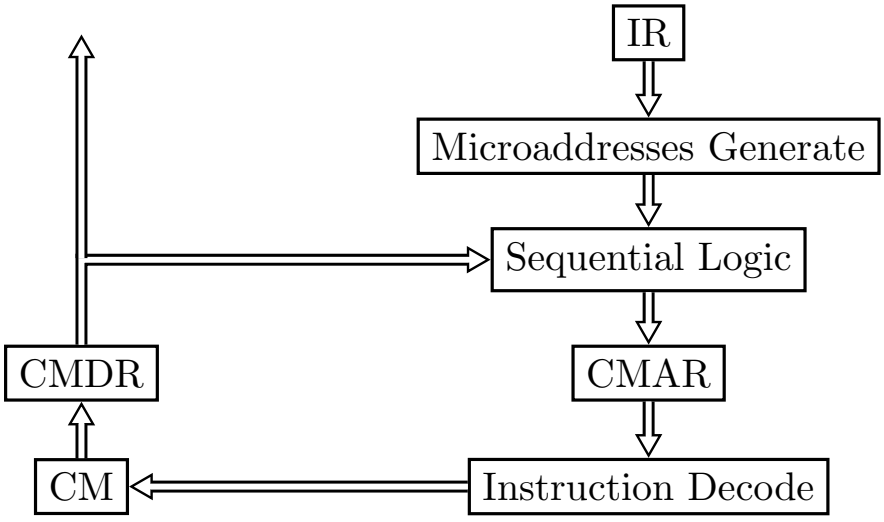

图 3.2: 微程序设计 CPU 框图

§4 组合逻辑设计

4.1 操作时间表

由于部分指令字长为 16 位，而机器字长和存储字长为 8 位，所以在取指周期，部分指令需要两次访存才能将指令完全取出。为使每条指令的机器周期数和每个机器周期的时钟周期数不变，二字长指令采用中央控制和局部控制相结合的方法，取指周期可以挪用间址周期的部分时钟周期。

另一方面，因为存储器读操作必须满足地址的建立和保持时间要求，而存储器的写操作还必须满足待写入数据的建立和保持时间要求。一般来说，当向存储器读数据时，可以不用显式地给出读命令，但是在地址有效后需要等待一时钟周期才能得到目标数据，所以必要的时间需要在机器周期中插入空操作，即 NOP 操作。而在写操作时，必须显式地给出写命令，且地址、数据都有效后必须保持一时钟周期再撤销才能使数据写入目标存储单元。

具体的操作时间表见附录。

4.2 逻辑表达式

1. M(PC)→MDR
   $= \text{FE} \cdot T_0 + \text{FE} \cdot T_1$
   $(\text{mov}_0+\text{mov}_1+\text{mov}_2+\text{mov}_3+\text{add}_0+\text{add}_1+\text{add}_2+$
   $\text{sub}_0+\text{sub}_1+\text{sub}_2+\text{and}_0+\text{and}_1+\text{and}_2+\text{or}_0+\text{or}_1+\text{or}_2+\text{jmp})$

2. MDR→IR
   $= \text{FE} \cdot T_0 + \text{FE} \cdot T_1$
   $(\text{mov}_0+\text{mov}_1+\text{mov}_2+\text{mov}_3+\text{add}_0+\text{add}_1+\text{add}_2+$
   $\text{sub}_0+\text{sub}_1+\text{sub}_2+\text{and}_0+\text{and}_1+\text{and}_2+\text{or}_0+\text{or}_1+\text{or}_2+\text{jmp})$

3. OP(IR)→IS,I,M
   $= \text{FE} \cdot T_0$

4. Ad(IR)→RADD1
   $= \text{FE} \cdot T_0$
   $(\text{mov}_0+\text{mov}_1+\text{mov}_2+\text{mov}_3+\text{add}_0+\text{add}_1+\text{add}_2+$

sub_0+sub_1+sub_2+ and_0+and_1+and_2+or_0+or_1+or_2)

5. PC+1→PC
   = FE $\cdot T_0$
   (mov_0+mov_1+mov_2+mov_3+add_0+add_1+add_2+
   sub_0+sub_1+sub_2+ and_0+and_1+and_2+or_0+or_1+or_2+jmp)
   +EX $\cdot T_2$
   (mov_0+mov_1+mov_2+mov_3+add_0+add_1+add_2+
   sub_0+sub_1+sub_2+ and_0+and_1+and_2+or_0+or_1+or_2+jmp)

6. NOP
   = FE $\cdot T_1$ + IND $\cdot T_0$

7. OP(IR)→ACC
   = FE $\cdot T_2$

8. Ad(IR)→RADD2
   = FE $\cdot T_2$ (mov_0+ add_0+sub_0+and_0+or_0)

9. Ad(IR)→MAR
   = IND $\cdot T_2$ (mov_1+mov_2+ add_1+sub_1+and_1+or_1)

10. M(MAR)→MDR
    = IND $\cdot T_1$ (mov_1+mov_2+ add_1+sub_1+and_1+or_1)

11. OP(MDR)→ACC
    = IND $\cdot T_1$ (mov_1+mov_2+ add_1+sub_1+and_1+or_1)

12. M(RADD2)→ACC
    = EX $\cdot T_0$ (mov_0+ add_0+sub_0+and_0+or_0)

13. ACC→M(RDD1)
    = EX $\cdot T_1$ (mov_0+mov_1+mov_2+mov_3)

14. M(RDD1)+ACC→M(RADD1)
    = EX $\cdot T_1$ (add_0+add_1+add_2)

15. M(RDD1)-ACC→M(RADD1)
    = EX ·T_1 (sub_0+sub_1+sub_2)

16. M(RDD1)&ACC→M(RADD1)
    = EX ·T_1 (and_0+and_1+and_2)

17. M(RDD1)ACC→M(RADD1)
    = EX ·T_1 (or_0+or_1+or_2)

18. ¬M(RDD1)→M(RADD1)
    = EX ·T_1 ·not

19. ACC→PC
    = EX ·T_2 ·jmp

注：

1. 指令助记符后的数字代表不同操作数类型的同种指令，顺序按上文列出时的顺序标号。
2. ¬ACC 代表 ACC 按位取反

§5 微程序设计

5.1 字符含义

字符	含义
PC	程序计数器
IR	指令寄存器
MAR	存储器地址寄存器
MDR	存储器数据寄存器
CMAR	控制存储器地址寄存器
CMDR	控制存储器数据寄存器
Ad	取地址
OP	指令译码
R	读信号

表 5.1: 字符含义

5.2 机器指令的微操作及节拍安排

5.2.1 取指周期的微操作及节拍安排

T_0 PC→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR,(PC)+1→PC
T_3 Ad(CMDR)→CMAR
T_4 MDR→IR,OP(IR)→ 微地址形成部件
T_5 OP(IR)→ 微地址形成部件 →CMAR

5.2.2 间址周期的微操作及节拍安排

T_0 Ad(IR)→MAR
T_1 M(MAR)→MDR
T_2 MDR→ACC
T_3 OP(IR)→ 微地址形成部件
T_4 OP(IR)→ 微地址形成部件 →CMAR

5.2.3 mov 指令

T_0 Ad(IR)→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR
T_3 Ad(CMDR)→CMAR
T_4 MDR→AC
T_5 Ad(CMDR)→CMAR, 取指微程序入口地址 →CMAR

5.2.4 add 指令

T_0 Ad(IR)→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR 注：若为 reg-mem 型 add 指令，则无 T_2。
T_3 Ad(CMDR)→CMAR
T_4 AC+MDR→AC 注：若为 reg-mem 型 add 指令，则该节拍
T_5 Ad(CMDR)→CMAR, 取指微程序入口地址 →CMAR

5.2.5 sub 指令

T_0 Ad(IR)→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR
T_3 Ad(CMDR)→CMAR
T_4 AC-MDR→AC
T_5 Ad(CMDR)→CMAR, 取指微程序入口地址 →CMAR

5.2.6 and 指令

T_0 Ad(IR)→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR
T_3 Ad(CMDR)→CMAR
T_4 AC&MDR→AC
T_5 Ad(CMDR)→CMAR, 取指微程序入口地址 →CMAR

5.2.7 or 指令

T_0 Ad(IR)→MAR,1→R
T_1 Ad(CMDR)→CMAR
T_2 M(MAR)→MDR
T_3 Ad(CMDR)→CMAR
T_4 ACMDR→AC
T_5 Ad(CMDR)→CMAR, 取指微程序入口地址 →CMAR

5.2.8 not 指令

T_0 Ad(IR)→AC
T_1 AC→AC

5.2.9 jmp 指令

T_0 Ad(IR)→PC

5.2.10 hlt 指令

T_0 设置运行状态位为零

5.3 微指令格式设计

5.3.1 操作字段含义

操作字段位数	含义
0	PC→MAR
1	M(MAR)→MDR
2	PC+1→PC
3	MDR→IR
4	MDR→ACC
5	Ad(IR)→MAR
6	ACC→MDR
7	1→W
8	MDR→M(MAR)
9	Ad(IR)→PC
10	Ad(IR)→RADD
11	ACC+RI→ACC
12	ACC-RI→ACC
13	ACC&RI→ACC
14	ACC|RI→ACC
15	ACC→ACC
16	ACC→PC
17	ACC→RI
18	0→S

表 5.2: 操作字段含义

5.3.2 微指令格式与微程序设计

因为微操作数量较少，且大多是所有机器指令所共有的微操作，所以微指令操作控制字段采用水平型微指令中的直接编码方式；顺序控制字段采用断定方式，由下地址直接给出。

微程序名称	微指令地址（十六进制）	微指令（二进制代码）																								
		操作控制字段																			顺序字段				
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
取指	0	1	1	1	1																	x	x	x	x	x
间址	1		1				1																			
mov0	2											1	1							1						
mov1	3					1						1								1						
mov2	4							1	1	1		1	1													
mov3	5					1						1								1						
add0	6											1	1	1						1						
add1	7					1						1		1						1						
add2	8					1						1		1						1						
sub0	9											1	1		1					1						
sub1	10					1						1			1					1						
sub2	11					1						1			1					1						
and0	12											1	1			1				1						
and1	13					1						1				1				1						
and2	14					1						1				1				1						
or0	15											1	1				1			1						
or1	16					1						1					1			1						
or2	17					1						1					1			1						
not	18											1	1					1		1						
jmp	19					1													1							
hlt	20																				1					

§6 CPU 的 Verilog 实现及 Quartus II 仿真

6.1 CPU 的硬件设置

1. CPU 字长为 8 位，8 个程序员可见的寄存器，分别命名为 $r_0, \cdots, r_7$。
2. 地址总线、数据总线各为 8 位，可访问 2^8 字节的地址空间。
3. 存储器由 ROM 和 RAM 组成，出于简便，系统工作区（ROM），用户工作区（RAM）分开编址。对于 RAM，读操作时无需显式给出读命令，但在地址有效后需要等待一个周期才能获得数据；写操作时需要显式给出写命令（WE），且要满足地址、数据的建立与保持时间要求。
4. 采用 1 字节或者 2 字节变长指令字，操作码采用定长格式。二字长指令在取指阶段需要访问两次内存。
5. 设置指令标记寄存器，存储当前运行指令的类别。
6. 采用定长三级时序，每个指令周期包含 3 个机器周期（取指周期、间址周期和执行周期），每个机器周期由 3 个节拍构成。用节拍（beat）控制指令的运行
7. 设置程序计数器（PC），指令寄存器（IR），存储器地址寄存器（MAR），存储器数据寄存器（MDR），累加器（ACC），寄存器地址寄存器（RADD，用来存储寄存器型指令中寄存器的地址）。

6.2 主要模块的 Verilog 实现

6.2.1 取指周期的指令译码操作

```
1          case (q_w[7:5])
2          3'b000: begin//mov
3                          case(q_w[4:3])
4                          2'b00:  begin//mov reg-reg
5                                          instruction <= 5'b00000;
6                                          radd1 <= q_w[2:0];
7                                          pc <= pc+1;
8                                          jp <= 2;
9                                          end
10                         2'b01:  begin//mov reg-mem;
11                                         instruction <= 5'b00001;
12                                         radd1 <= q_w[2:0];
```

```
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        2'b10:  begin//mov mem-reg;
                                    instruction <= 5'b00010;
                                    radd2 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        2'b11:  begin//mov reg-立即数;
                                    instruction <= 5'b00011;
                                    radd1 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        endcase
                        end
            3'b001: begin//add
                        case(q_w[4:3])
                        2'b00:  begin//add reg-reg;
                                    instruction <= 5'b00100;
                                    radd1 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        2'b01:  begin//add reg-mem;
                                    instruction <= 5'b00101;
                                    radd1 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        2'b10:  begin//add reg-立即数;
                                    instruction <= 5'b00110;
                                    radd1 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        endcase
                        end
            3'b010: begin//sub
                        case(q_w[4:3])
                        2'b00:  begin//sub reg-reg;
                                    instruction <= 5'b01000;
                                    radd1 <= q_w[2:0];
                                    pc <= pc+1;
                                    jp <= 2;
                                    end
                        2'b01:  begin//sub reg-mem;
```

```
61                                      instruction <= 5'b01001;
62                                      radd1 <= q_w[2:0];
63                                      pc <= pc+1;
64                                      jp <= 2;
65                                      end
66                      2'b10:  begin//sub reg—立即数;
67                                      instruction <= 5'b01010;
68                                      radd1 <= q_w[2:0];
69                                      pc <= pc+1;
70                                      jp <= 2;
71                                      end
72                      endcase
73                      end
74          3'b011: begin//and
75                      case(q_w[4:3])
76                      2'b00:  begin//and reg—reg;
77                                      instruction <= 5'b01100;
78                                      radd1 <= q_w[2:0];
79                                      pc <= pc+1;
80                                      jp <= 2;
81                                      end
82                      2'b01:  begin//and reg—mem;
83                                      instruction <= 5'b01101;
84                                      radd1 <= q_w[2:0];
85                                      pc <= pc+1;
86                                      jp <= 2;
87                                      end
88                      2'b10:  begin//and reg—立即数;
89                                      instruction <= 5'b01110;
90                                      radd1 <= q_w[2:0];
91                                      pc <= pc+1;
92                                      jp <= 2;
93                                      end
94                      endcase
95                      end
96          3'b100: begin//or
97                      case(q_w[4:3])
98                      2'b00:  begin//or reg—reg;
99                                      instruction <= 5'b10000;
100                                     radd1 <= q_w[2:0];
101                                     pc <= pc+1;
102                                     jp <= 2;
103                                     end
104                     2'b01:  begin//or reg—mem;
105                                     instruction <= 5'b10001;
106                                     radd1 <= q_w[2:0];
107                                     pc <= pc+1;
108                                     jp <= 2;
```

```
                                        end
                        2'b10:  begin//or reg—立即数;
                                        instruction <= 5'b10010;
                                        radd1 <= q_w[2:0];
                                        pc <= pc+1;
                                        jp <= 2;
                                        end
                        endcase
                        end
        3'b101: begin//not
                        instruction <= 5'b10100;
                        radd1 <= q_w[2:0];
                                        jp <= 2;
                        end
        3'b110: begin//jmp
                                instruction <= 5'b11000;
                                pc <= pc + 1;
                                jp <= 2;
                                end
        3'b111: instruction <= 5'b11100;//hlt
        default:jp<= 2;
        endcase
```

Verilog 程序较长，为避冗杂，这里不再列出。另有源码附上。

6.3 Quartus II 仿真与测试

可以手动将汇编代码编译成该 CPU 可以识别的机器代码。由于缺少条件转移指令，无法实现循环和条件分支，这里我增加了一条 jnz 指令，当运算结果不为 0 是，程序跳转到目标代码处，跳转地址由指令直接给出。

```
mov r1,0;
mov r2,8;
for: add r1,1;
sub r2,1;
jz for;
hlt;
```

波形仿真结果在

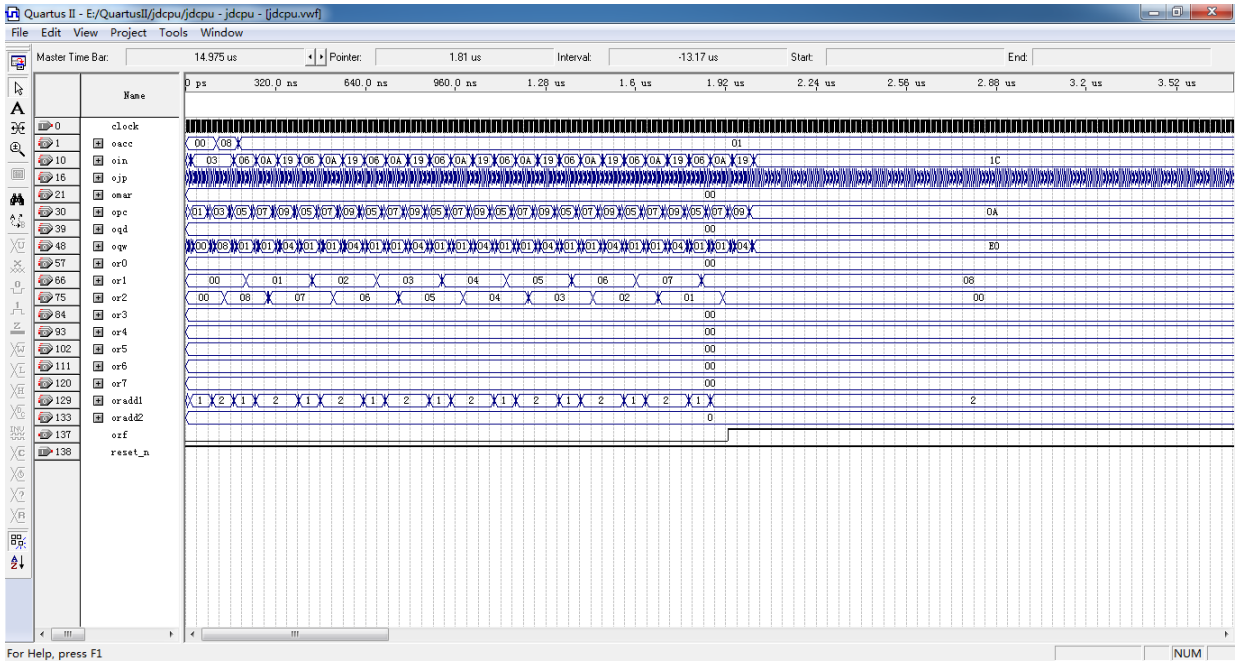
Quartus II - E:/QuartusII/jdcpu/jdcpu - jdcpu - [jdcpu.vwf]
File Edit View Project Tools Window
Master Time Bar: 14.975 us Pointer: 1.81 us Interval: -13.17 us Start: End:
Name
clock
oacc
oin
ojp
omar
opc
oqd
oqw
or0
or1
or2
or3
or4
or5
or6
or7
oradd1
oradd2
orf
reset_n
For Help, press F1
NUM

图 6.1: 仿真输出波形

§7 附录

7.1 操作时间表

工作周期标记	节拍	状态条件	微操作命令信号	mov_0	mov_1	mov_2	mov_3	add_0	add_1	add_2	sub_0	sub_1	sub_2	and_0	and_1	and_2	or_0	or_1	or_2	not	jmp	hlt
FE(取指)	T0		M(PC)→MDR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
			MDR→IR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
			OP(IR)→IS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
			Ad(IR)→RADD1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
			PC+1→PC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1	
	T1		NOP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	T2		M(PC)→MDR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1	
			MDR→IR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1	
			OP(IR)→ACC				1			1			1			1			1		1	
			Ad(IR)→RADD2	1				1			1			1			1					
			Ad(IR)→MAR		1	1			1			1			1			1				
IND(间址)	T0		NOP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	T1		M(MAR)→MDR		1	1			1			1			1			1				
			OP(MDR)→ACC		1	1			1			1			1			1				
	T2		NOP	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EX(执行)	T0		M(RADD2)→ACC	1				1			1			1			1					
	T1		ACC→M(RDD1)	1	1	1	1															
			M(RDD1)+ACC→M(RADD1)					1	1	1												
			M(RDD1)-ACC→M(RADD1)								1	1	1									
			M(RDD1)&ACC→M(RADD1)											1	1	1						
			M(RDD1)|ACC→M(RADD1)														1	1	1			
			¬M(RDD1)→M(RADD1)																	1		
	T2		ACC→PC																		1	
			PC+1→PC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	