拓展实验报告

微程序的控制单元实现

姓　　名：杨尚斌

导　　师：刘宏伟教授

学　　科：计算机科学与技术

学　　号：1130310128

摘　要

计算机之所以能够自动协调的工作，是由于控制单元（CU）的统一指挥。因此 CU 的实现在计算机中是很重要的一部分。

控制单元大体上有两种设计方式，一种是组合逻辑设计，一种是微程序设计。

本文主要讨论的是微程序设计的一些实现。微程序设计思想是英国剑桥大学的 M.V.Wilkes 在 1951 年的时候提出的，但由于当时的一些限制，这个微程序设计的思想一直没有实现，知道后面 60 年代的时候半导体存储器的出现，才使得这个思想成为现实。

相比于组合逻辑设计，微程序设计省去了组合逻辑设计中对于逻辑表达式的化简结构，不需要考虑逻辑门级数和门的扇入系数，使得整个设计更加简单。并且由于控制信号是以二进制的代码形式出现的，因此只需要修改微指令的代码，就可以改变操作内容，便于调试，修改，甚至对机器指令进行增删，极大的利于计算机的防真。

关键词： 微程序 微指令 CU

Abstract

The computer was able to automatically coordinate the work, is due to the control unit (CU) is the unified command. Therefore CU is implemented in a computer is a very important part of it.
Control unit design in general, there are two ways, one is a combination of logic design, one is micro programming.
This article discusses some implementations micro programming
Micro-programming idea is MVWilkes Cambridge University in 1951, when presented, but due to some restrictions at the time, the idea of micro-programming has not been achieved, knowing that appear after the 1960s, when a semiconductor memory, which makes this Thoughts become reality.
Compared to the combinational logic design, eliminating the need for a combination of micro-program designed to simplify the structure of the logical design of the logical expression, without regard to logic gates and fan series coefficients of the door, making the whole design easier. And since the control signal is in the form of binary code that appears, and therefore only need to modify microinstruction code, you can change the operation content, easy to debug, modify, and even machine instructions deletions, greatly beneficial anti-truth machine.

Keywords: micro program、 micro instruction 、control unit

1、实验的目的与意义

在微程序设计中，将一条机器指令编写成一个微程序，每一个微程序包含若干条微指令，每一条微指令对应一个或者几个微操作的命令。然后把这些微程序存到一个控制存储器中，用寻找用户程序机器指令的方法来寻找每个微程序的微指令。由于这些微指令是以二进制的代码形式表示的，每位代表一个控制信号，因此，逐一执行每一条微指令，也就是相应的完成了一条机器指令的全部操作。

因此，微程序控制单元的核心就是一个控制存储器。本实验主要用 VHDL 来模拟具有 10 条指令的 CU。

2、实验的设计与分析

实验准备：

1、对应机器指令的微操作以及节拍安排

1）、取指阶段

T0 PC->MAR, 1->R

T1 M(MAR)->MDR, (PC)+1->PC

T2 MDR->IR, OP(IR)->微地址形成部件

2）、执行阶段

1. CLA 指令

T0 0->AC

2.COM 指令

T0 ～AC->AC

3.SHR 指令

T0 L (AC) -> R (AC), AC0->AC0

4.CSL 指令

T0 R (AC) -> L (AC), AC0 ->ACn

5.STP 指令

T0 0->G

6.ADD 指令

T0 Ad (IR) -> MAR, 1->R

T1 M (MAR) -> MDR

T2　　(AC) + (MDR) -> AC

7.STA 指令

T0　　Ad(IR)->MAR,　1->W

T1　　AC->MDR

T2　　MDR->M(MAR)

8.LDA 指令

T0　　Ad(IR)->MAR,　1->R

T1　　M(MAR)->MDRT2MDR->AC

9.JMP 指令

T0　　Ad(IR)->PC

10.BAN 指令

T0　　A0*Ad(IR)+~A0*(PC)->PC

2、 确定微指令的格式

1）0-17 (指令采用 18 个操作控制位表示操作控制字段，按照下面的方式进行编码)

第 0 位表示控制 PC->MAR

第 1 位表示控制 1->R

第 2 位表示控制 M(MAR)->MDR

第 3 位表示控制 (PC)+1->PC

第 4 位表示控制 MDR->IR

第 5 位表示控制 0->AC

第 6 位表示控制～AC->AC

第 7 位表示控制 L(AC)->R(AC), AC0->AC0

第 8 位表示控制 R(AC)->L(AC), AC0->ACn

第 9 位表示控制 0->G

第 10 位表示控制 Ad(IR)->MAR

第 11 位表示控制 (AC)+(MDR)->AC

第 12 位表示控制 1->W

第 13 位表示控制 AC->MDR

第 14 位表示控制 MDR->M(MAR)

第 15 位表示控制 MDR->AC

第 16 位表示控制 Ad(IR)->PC

第 17 位表示控制 A0*Ad(IR)+~A0*(PC)->PC

2）18

第 18 位控制下地址的形成方式，如果为 1 代表采用指令的操作编码方式，如果为 0 表示采用微指令的下地址字段的编码方式。可使用一个简单的二路选择器来实现。

3）19-23

如果有下地址的话代表下地址，即顺序控制字段。

3、 指令系统

在这个 CU 中，按照下面的方式对操作符进行相应的编码

CLA		02H	1；累加器清零
COM		03H	1；累加器取反
SHR		04H	1；算数右移
CSL		05H	1；循环左移
STP		06H	1；停机
ADD	[*]	07H	3；加法
STA	[*]	09H	3；存数
LDA	[*]	0BH	3；取数
JMP	*	0DH	1；无条件跳转
BAN	*	0FH	1；负则转

实验分析：

按照上面的一些约定，我们可以得出下面的这个表格（注：微指令地址我们约定用指令码左移一位得到）

微程序名称	微指令地址	微指令（二进制代码）																						
		操作控制字段																		顺序控制字段				
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
取指	00H	1	1																						1
	01H			1	1																			1	
	02H					1														1	X	X	X	X	X
CLA	04H						1																		
COM	06H							1																	

SHR	08H							1															
CSL	0AH								1														
STP	0CH									1													
ADD	0EH		1								1									1	1	1	1
	0FH			1														1					
	10H											1											
STA	12H										1		1					1			1	1	
	13H													1				1		1			
	14H														1								
LDA	16H		1								1							1		1	1	1	
	17H			1														1	1				
	18H															1							
JMP	1AH																1						
BAN	1CH																	1					

实验内容：

在整个用 VHDL 模拟 CU 的过程中，我们把整个过程分为四个部分进行实现，分别标记为 convertaddr、MUX、ROM.其作用分别为：convertaddr-将指令码左移一位得到微指令地址；MUX-判断微指令中第 18 位的值，并且判断下一个地址是谁；ROM-按照上图中的约定对每个微指令地址记录其对应的二进制微指令；Splitcode-将 24 位的微指令地址进行划分，得到最后所需要的 0-17 位操作控制字段。实验代码具体分析如下：

总模块：

```
entity CU is
    Port ( clk : in  STD_LOGIC;
           op_code : in  STD_LOGIC_VECTOR (4 downto 0);
           ctrl_signal : out  STD_LOGIC_VECTOR (17 downto 0));end CU;
```

输入一个时钟信号和指令码，输出的是这个指令码对应的数条微指令。其模块由四个 component 组成：

```
architecture Behavioral of CU is
component Splitcode
```

```
    Port (
        clock   : IN Std_logic;
        u_op : in  STD_LOGIC_VECTOR (0 TO 23);
        control : out  STD_LOGIC_VECTOR (0 TO 17);
        mode_sel: out  STD_LOGIC;
        next_add : out  STD_LOGIC_VECTOR (4 DOWNTO 0));
end component;
component convertaddr
   Port (
        op : in  STD_LOGIC_VECTOR (4 downto 0);
     op_add : out  STD_LOGIC_VECTOR (4 downto 0));
end component;
component MUX
    PORT(
        mode:IN STD_LOGIC;
        next_add:IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        op_addr  :IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        out_add :OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
end component;
component ROM
   Port (
        add : in  STD_LOGIC_VECTOR (4 downto 0);
     data_out : out  STD_LOGIC_VECTOR (0 to 23));
end component;

signal op_add_MUX:std_logic_vector(4 downto 0);
signal mode_MUX  :std_logic;
signal next_add_MUX:std_logic_vector(4 downto 0);
signal MUX_CM  :std_logic_vector(4 downto 0);
signal CM_CMAR   :std_logic_vector(0 to 23);

begin
unit1:convertaddr port map(op_code, op_add_MUX);
```

```
    unit2:MUX port map(mode_MUX, next_add_MUX, op_add_MUX, MUX_CM);
    unit3:ROM port map(MUX_CM, CM_CMAR);
    unit4:Splitcode port map(clk, CM_CMAR, ctrl_signal, mode_MUX,
next_add_MUX);
    end Behavioral;
```

Convertaddr 模块：

```
    --convertaddr
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL,IEEE.NUMERIC_STD.ALL;
    entity convertaddr is
       Port ( op : in  STD_LOGIC_VECTOR (4 downto 0);
              op_add : out  STD_LOGIC_VECTOR (4 downto 0));
    end convertaddr;

    architecture Behavioral of convertaddr is
    begin
        process(op)
            begin
                op_add <= op(3 downto 0) & '0';
            end process;
    end Behavioral;
```

输入是指令码，其主要操作是对他进行左移操作，然后转化为微指令地址进行输出。

MUX 模块

```
    --MUX
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL,IEEE.NUMERIC_STD.ALL;
    entity MUX is
```

```
PORT(
    mode:IN STD_LOGIC;
    next_add:IN STD_LOGIC_VECTOR(4 DOWNTO 0);
    op_addr :IN STD_LOGIC_VECTOR(4 DOWNTO 0);
    out_add :OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
end MUX;


architecture Behavioral of MUX is
begin
    process(mode,next_add,op_addr)
    begin
        case mode is
            when'0' => out_add<=next_add;
            when others => out_add<=op_addr;
        end case;
    end process;
end Behavioral;
```

根据输入的 mode 判断下一个地址是什么，赋值给 out_add 进行输出。

ROM 模块

```
--ROM
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
entity ROM is
   Port ( add : in  STD_LOGIC_VECTOR (4 downto 0);
          data_out : out  STD_LOGIC_VECTOR (0 to 23));
end ROM;

architecture Behavioral of ROM is
type microcode_array is array(28 downto 0) of std_logic_vector(0 to 23);
```

```
    constant code : microcode_array:=(
    0=> "110000000000000000000001", 1=> "001100000000000000000010", 2=>
"000010000000000001UUUUU", 4=> "000001000000000000000000",
    6=> "000000100000000000000000", 8=> "000000010000000000000000", 10=>
"000000001000000000000000", 12=> "000000000100000000000000",
    14=> "010000000010000000001111", 15=> "001000000000000000010000", 16=>
"000000000001000000000000",18=> "000000000010100000010011",
    19=> "000000000000010000010100",20=> "000000000000001000000000", 22=>
"010000000010000000010111", 23=> "001000000000000000011000",
    24=> "000000000000000100000000", 26=> "000000000000000010000000", 28=>
"000000000000000001000000",others=>"000000000000000000000000");
    begin
        data_out <= code(conv_integer(add));
    end Behavioral;
```

根据前面的相关约定确定每条微指令地址对应的微指令二进制代码代码

Splitcode:

```
    --Splitcode
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    entity Splitcode is
    Port (
        clock   : IN Std_logic;
        u_op : in  STD_LOGIC_VECTOR (0 TO 23);
        control : out  STD_LOGIC_VECTOR (0 TO 17);
        mode_sel: out  STD_LOGIC;
        next_add : out  STD_LOGIC_VECTOR (4 DOWNTO 0));
    end Splitcode;
```

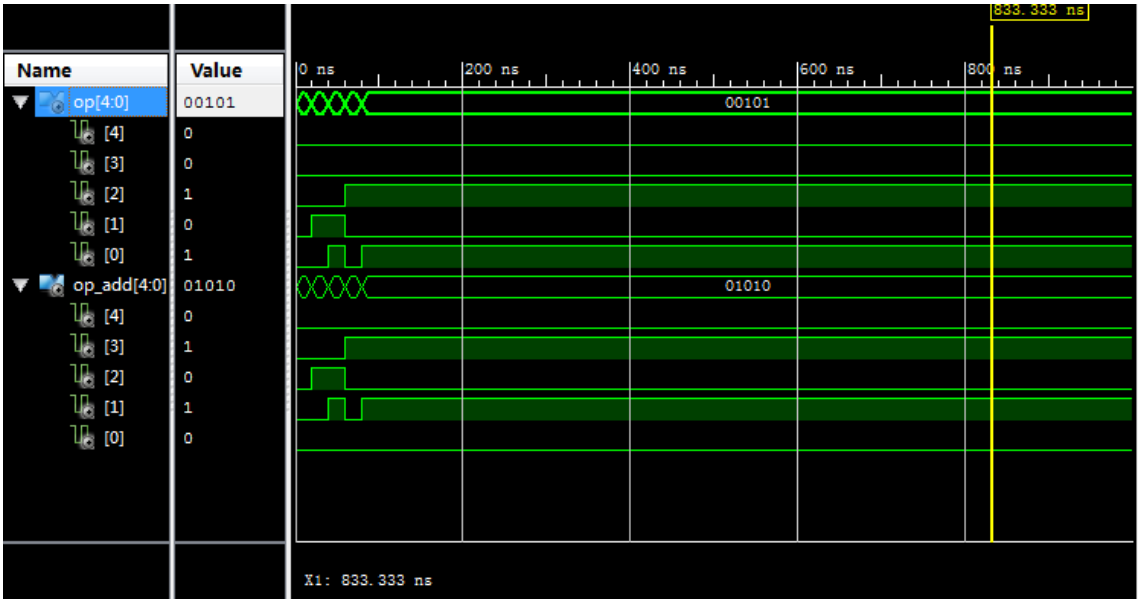
```
architecture Behavioral of Splitcode is
SIGNAL int_reg : Std_logic_vector(0 TO 23);
BEGIN
    main_proc : PROCESS
    BEGIN
        WAIT UNTIL falling_edge(clock);
        int_reg <= u_op;
    END PROCESS;
    control <= int_reg(0 TO 17);
    mode_sel<= int_reg(18);
    next_add <= int_reg(19 TO 23);
end Behavioral;
```

对于上一个阶段产生的微指令二进制 24 位代码进行处理，得到 0-17 位的操作控制字段。

3、实验的仿真与分析

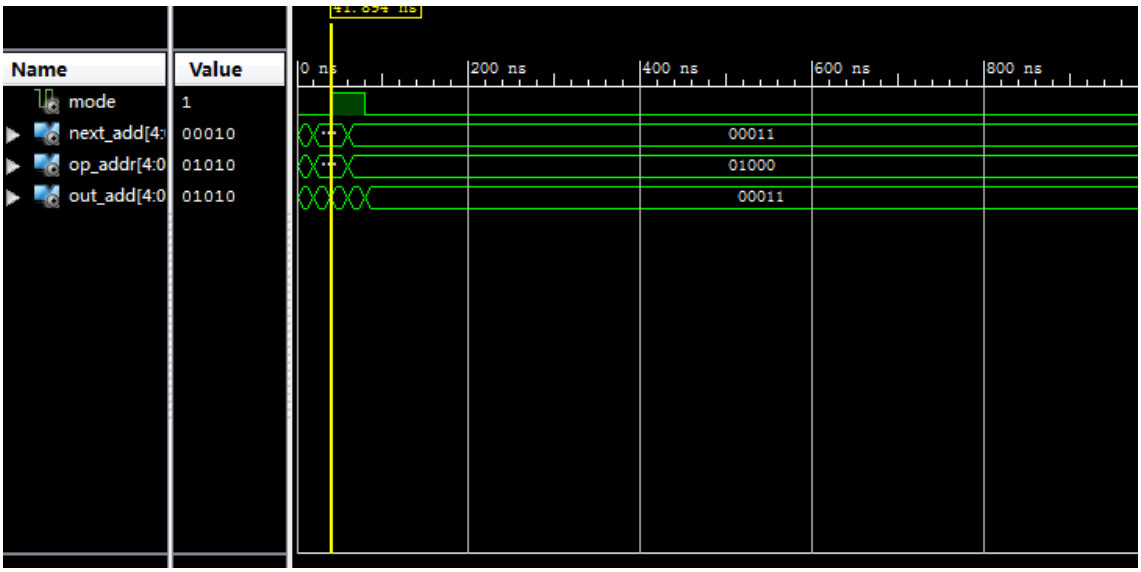
在仿真阶段，进行了多次的模拟，主要是对每个模块的操作进行了单元仿真测试。在最后确保无误之后对整个模块进行了仿真测试。

Convertaddr 模块测试



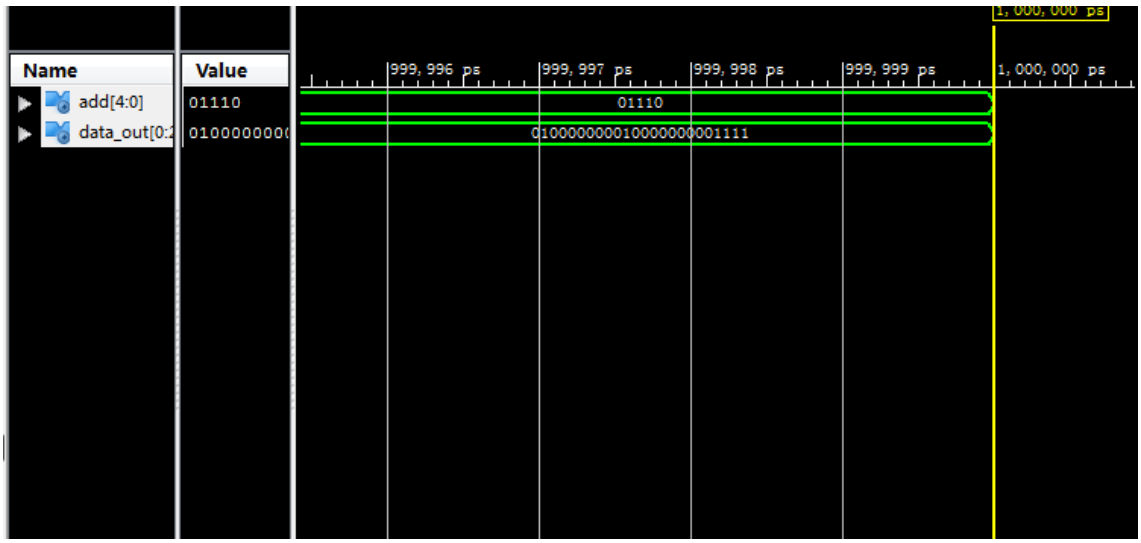
输入 00101 ，输出的是左移后的 01010

MUX 模块测试



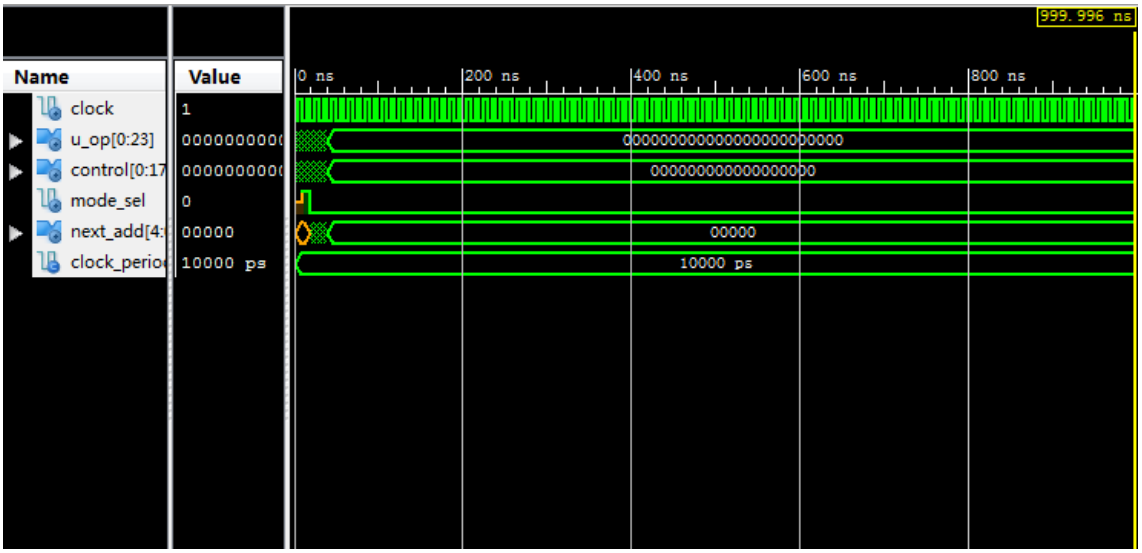
当 mode 是 1 的时候，输出 out_addr 和 op_addr 相同，当 mode 为 0 的时候，输出和 next_addr 下地址相同。

ROM 模块测试



输入是 01110 的情况下，输出的是 01110 这个微指令地址对应的微指令二进制代码。

Splitcode 模块测试



把 u_op 这个微指令二进制代码划分成 0-18 19 20 -13 四个部分。

总的 CU 模块

如图，当输入微指令码 op_code 的时候输出对应的微指令二进制代码。例如在图中的一段中，输出的微指令二进制代码为 00011(03H COM 操作，则对应的微指令二进制代码依次为：110000000000000000

00110000000000000 0000100000000000000 0000001000000000000 符合上面得出的表格。

4、实验的拓展与调研

1）、静态微程序设计和动态微程序设计

通常指令系统是固定的，对应每一条机器指令的微程序是计算机设计者事先编好的，因此一般微程序无需改变，这种微程序设计技术即称为静态微程序设计，其控制存储器采用 ROM，我们在教材中用到的就是这种静态的微程序设计。

如果采用 EPROM 作为控制存储器，人们可以通过改变微指令和微程序来改变机器的质量系统，这种微程序设计技术成为动态微程序设计。动态微程序设计由于可以根据需要改变微指令和微程序，因此可以在一台机器上上实现不同类型的指令系统，更加有利于仿真。但是这种设计对用户的要求恒高，目前比较难以推广。

2）、毫微程序设计

微程序可以看做是用来解释机器指令的，毫微程序可以看做是解释微程序的，而组成毫微程序的毫微指令则是用来解释微指令的。采用毫微程序设计计算机的有点是用少量的控制存储器空间来达到高度的并行。

一般来说，毫微程序设计采用两级微程序的设计方法。第一级微程序为垂直型微指令，并行功能不强，但是有严格的顺序结构，由他确定后续的微指令地址，当需要的时候可以调用第二级。第二级微程序为水平型微指令，具有很强的并行操作能力，但是不包含后续微指令的地址。第二级微程序执行完毕后又返回到第一级微程序。两级微程序分别放在两级控制存储器内。

参考文献

[1] 微程序控制器的设计与实现 杨波 高德远 计算机工程与应用（2001-04-01）

[2] 简论微程序控制器机理 周秋和 高等函数学报（自然科学版）（2002-02-25）

[3] 微程序的设计与实现 张雅茹刘凯歌 甘肃科技 （2009-11-08）

[4] VHDL 语言对微程序设计思想的描述及模拟应用 孙凌宇 冷明 井冈山师范学院学报 （2003-10-30）

[5] 计算机组成原理 唐朔飞 高等教育出版社 第二版