计算机组成原理大作业——设计实现控制单元 CU

1120310xxx xxxx

微程序设计控制单元的主要任务是编写对应各条机器指令的微程序，具体步骤是首先写出对应机器指令的全部微操作及节拍安排，然后确定微指令格式，最后编写出每条微指令的二进制代码（称为微指令码点）。

写出对应机器指令的微操作及节拍安排

不考虑简介寻址和中断的情况。下面分别按取指阶段和执行阶段列出其微操作序列。

（1）取指阶段的微操作及节拍安排

T0　　PC->MAR, 1->R
T1　　M(MAR)->MDR, (PC)+1->PC
T2　　MDR->IR, OP(IR)->微地址形成部件（编码器：指令码->微地址）**（此步为组合逻辑，自动完成，不需要控制信号）**

（2）执行阶段的微操作及节拍安排

执行阶段的微操作由操作码性质而定，同时也需要考虑下地址的形成问题。

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)->MDR
T2　　MDR->AC
9）JMP 指令
T0　　Ad(IR)->PC
10）BAN 指令
T0　　A0*Ad(IR)+~A0*(PC)->PC

确定微指令格式

微指令的格式包括微指令的编码方式、下地址的形成方式和指令字长 3 方面。

（1）微指令的编码方式

直接编码。具体如下表。
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）下地址形成方式

采用微指令的下地址字段和指令的操作码编码两种形成方式。设置一控制位，0 表示前一种方式，1 表示后一种方式，作为二路数据选择器的控制端输入。

第 18 位表示控制下地址形成方式 0 表示选择顺序控制字段 1 表示选择操作码编码结果

（3）微指令字长

假设模型机存储字长和指令字长均为 16 位，地址字长为 8 位。采用 16 位定长指令，操作码为前 5 位,中间 3 位保留，地址码为后 8 位。

指令系统如下：

```
；助记符        操作数          指令码          长度
；-------------------------------------------------------------------------
CLA                             02H             2；累加器清零
COM                             03H             2；累加器取反
SHR                             04H             2；算数右移
CSL                             05H             2；循环左移
STP                             06H             2；停机
ADD             [*]             07H             2；加法
STA             [*]             09H             2；存数
LDA             [*]             0BH             2；取数
JMP             *               0DH             2；无条件跳转
BAN             *               0EH             2；负则转
```

18 个微操作，19 条微指令。不考虑指令系统的扩充，操作控制字段取 18 位（0～17），下地址控制字段取 1 位（18），顺序控制字段（下地址）取 5 位（19～23）。

编写微指令码点

表中空格中“0”省略。

对应 10 条机器指令的微指令码点

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						

确定 CU 的输入输出

输入：OP 操作码，CLK 时钟信号

输出：18 位控制信号

确定 CU 各部件内部逻辑

（1）CMDR

24 位寄存器。

（2）微地址形成部件

5 位。操作码左移 1 位末位补零，形成微程序入口地址。

（3）多路选择

5 位 2 选 1 多路选择器，A18 为控制输入端。

（4）控制存储器

24 位×32 个存储单元的 ROM。附带 5-32 译码器。

学习用 VHDL 借助 Xilinx 软件实现 CU

学习流程大致如下：

（1）画出 CU 各部件连接图，同课本 418 页图 14.16。

（2）阅读博客，了解 VHDL 语言的功能、语法、设计思路，以及 Xilinx 的安装、使用。

（3）寻找 CU 各部件单独实现的代码，借助已有代码的语法结构，修改以实现自己需要的功能。对于过程中出现的错误，通过百度出的技术博客、英文原版的标准库说明、同学的经验，一一解决。

（4）学习编写 testbench，同样也是找相似的代码，读懂，仿照着写出自己的测试要求。

（5）元件调用。各模块编写完成后，通过百度学习元件调用的方法，从网上优秀课件中找到元件实例化的语法、用法，最终实现整个 CU 各模块的协同工作。

（6）整个过程都反复的在看 VHDL 的系统讲解，一开始没有实例，系统讲解不知所云何物，随着在实例中碰到各种问题，再回过头看理论讲解，才不断有所收获。大约反复 5～7 遍，可以大体上了解 VHDL 语言的各个要素以及编写思想。

代码及仿真波形

代码

（1）微地址形成部件

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity uaddr_gen is
    Port ( op : in  STD_LOGIC_VECTOR (4 downto 0);
           op_add : out  STD_LOGIC_VECTOR (4 downto 0));
end uaddr_gen;

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

（2）5 位 2 选 1 多路选择器

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL,IEEE.NUMERIC_STD.ALL;

entity MUX is
PORT(
        mode            :IN STD_LOGIC; --输入控制信号
        next_add :IN STD_LOGIC_VECTOR(4 DOWNTO 0); --5 位下地址数据端
        op_addr         :IN STD_LOGIC_VECTOR(4 DOWNTO 0); --5 位微程序入口地址数据端
        out_add  :OUT STD_LOGIC_VECTOR(4 DOWNTO 0)); --5 位下地址输出
end MUX;

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

（3）控制存储器

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.ALL;            --CONV_INTEGER

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=> "00000001000000000000000",
              10=> "00000001000000000000000",
              12=> "00000000100000000000000",
              14=> "01000000001000000001111",
              15=> "00100000000000000010000",
              16=> "00000000001000000000000",
              18=> "00000000010100000010011",
              19=> "00000000000001000010100",
              20=> "00000000000001000000000",
              22=> "01000000001000000010111",
              23=> "00100000000000000011000",
              24=> "00000000000000100000000",
              26=> "00000000000000010000000",
              28=> "00000000000000001000000",
           others=>"000000000000000000000000");
begin
        data_out <= code(conv_integer(add));
end Behavioral;
```

（4）CMDR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CMDR is
 GENERIC(n : Positive := 24); --size of counter/shifter
    Port (
          clock   : IN Std_logic; --serial inputs
          u_op : in  STD_LOGIC_VECTOR (0 TO (n-1));
    control : out  STD_LOGIC_VECTOR (0 TO 17);
          mode_sel: out  STD_LOGIC;
    next_add : out  STD_LOGIC_VECTOR (4 DOWNTO 0));
end CMDR;

architecture Behavioral of CMDR is
SIGNAL int_reg : Std_logic_vector(0 TO 23);

BEGIN
 main_proc : PROCESS
 BEGIN
 WAIT UNTIL rising_edge(clock);
 int_reg <= u_op;
 END PROCESS;
 --connect internal register to dataout port
 control <= int_reg(0 TO 17);
 mode_sel<= int_reg(18);
 next_add <= int_reg(19 TO 23);

end Behavioral;
```

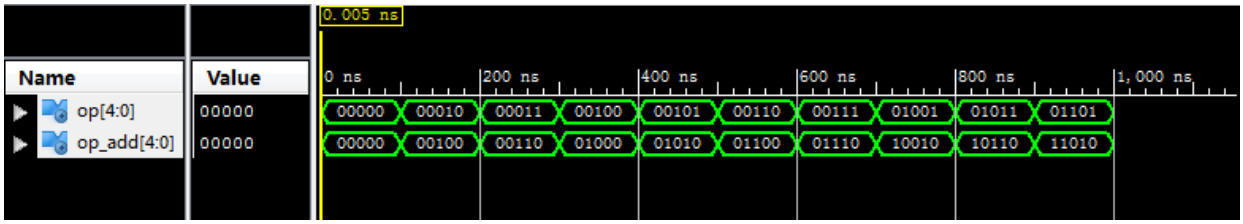
（5）总的CU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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;

architecture Behavioral of CU is

component CMAR
 GENERIC(n : Positive := 24); --size of counter/shifter
    Port (
          clock   : IN Std_logic; --serial inputs
          u_op : in  STD_LOGIC_VECTOR (0 TO (n-1));
    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 uaddr_gen
    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; --输入控制信号
        next_add :IN STD_LOGIC_VECTOR(4 DOWNTO 0); --5位下地址数据端
        op_addr  :IN STD_LOGIC_VECTOR(4 DOWNTO 0); --5位微程序入口地址数据端
```

```
        out_add  :OUT STD_LOGIC_VECTOR(4 DOWNTO 0)); --5位下地址输出
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
u1:uaddr_gen port map(op_code, op_add_MUX);
u2:MUX            port map(mode_MUX, next_add_MUX, op_add_MUX, MUX_CM);
u3:ROM       port map(MUX_CM, CM_CMAR);
u4:CMAR      port map(clk, CM_CMAR, ctrl_signal, mode_MUX, next_add_MUX);
end Behavioral;
```

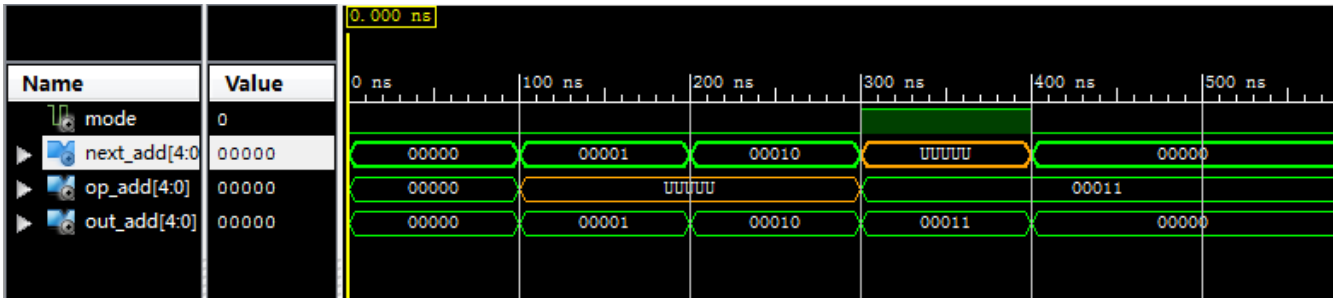
仿真波形

（1）微地址形成部件



可以看出，op_add 是 op 左移一位的结果。

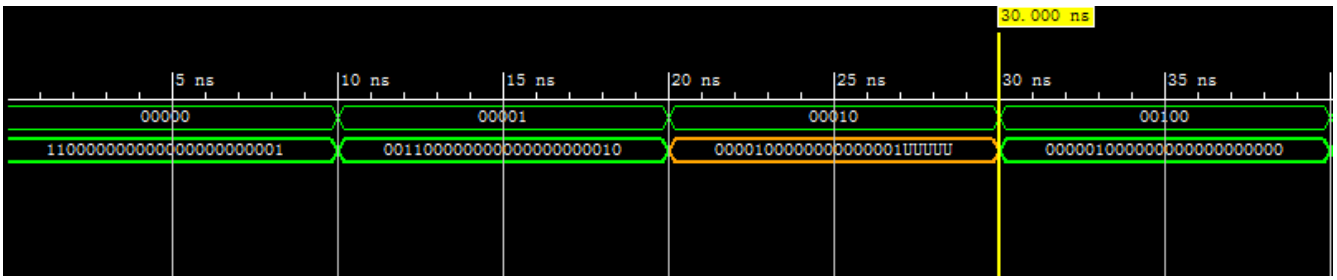
（2）5 位 2 选 1 多路选择器



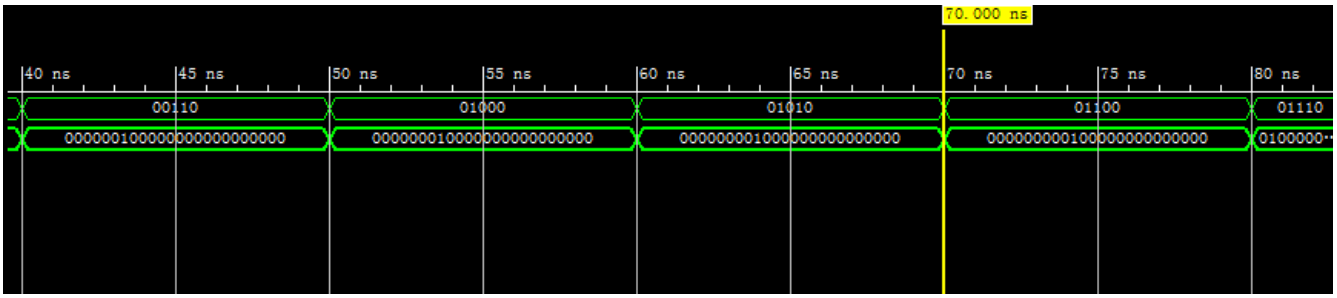
当 mode 为 0 时，out_add 输出 next_add，当 mode 为 0 时， out_add 输出 op_add。

（3）控制存储器

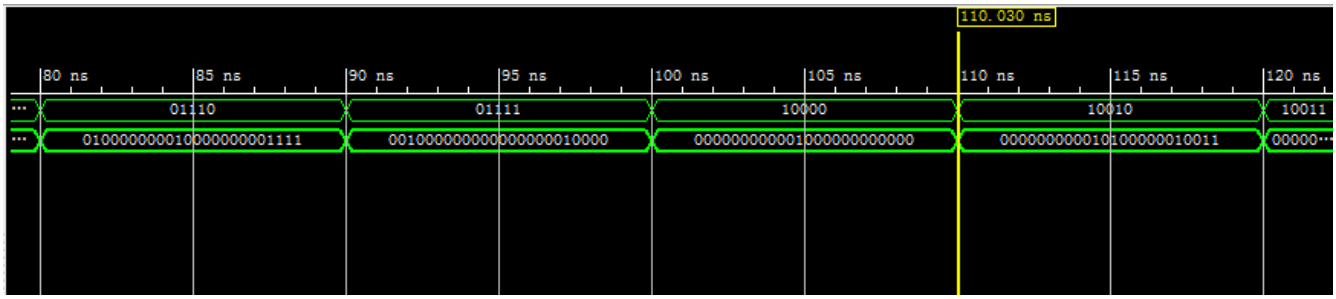
下图一一输出每条微指令。



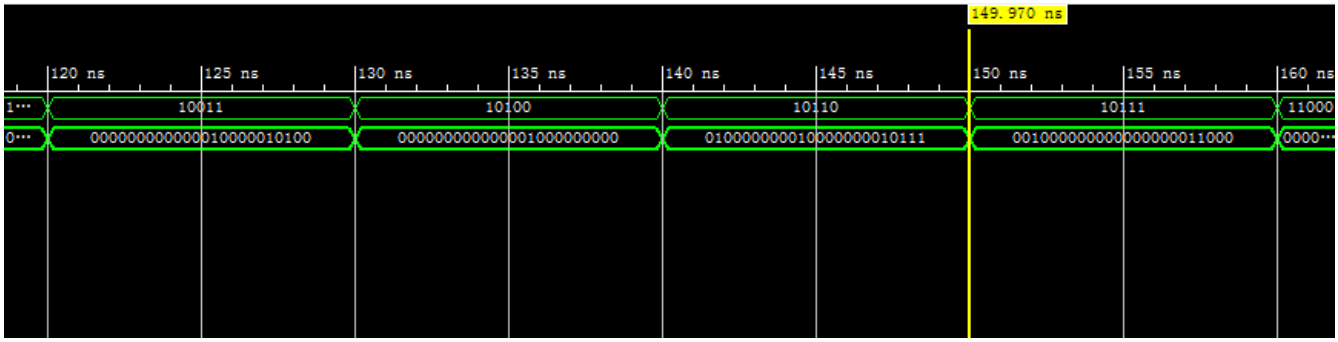
00H　　01H　　02H　　CLA



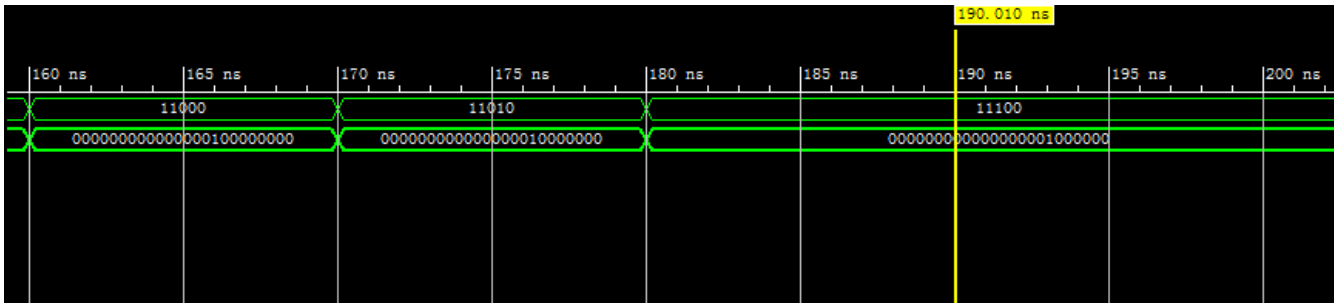
COM SHR CSL STP



ADD STA

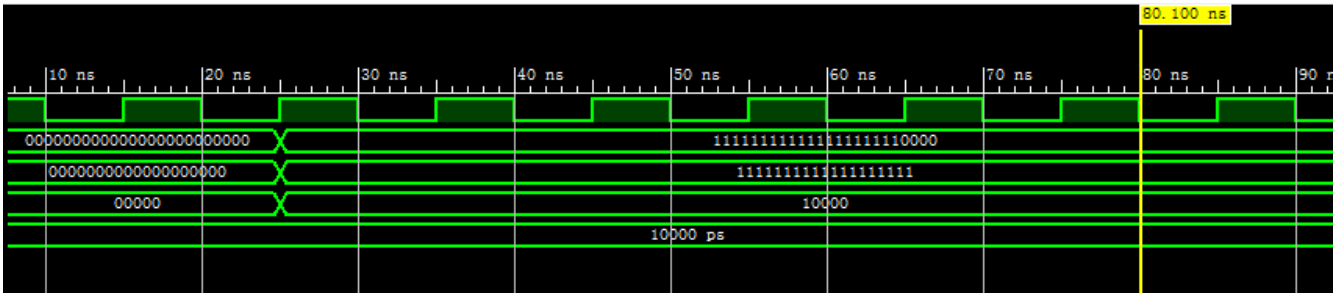


LDA



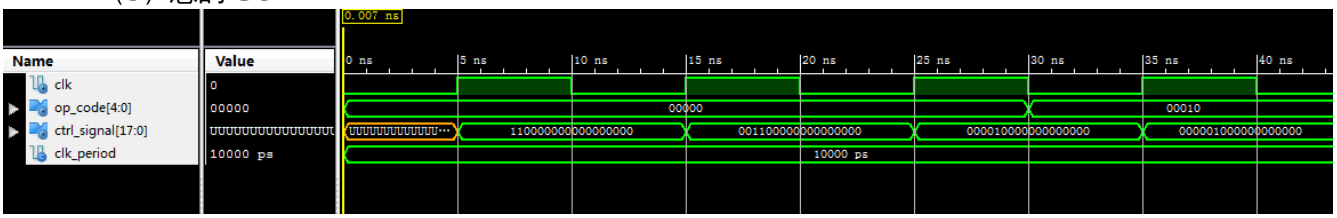
JMP BAN

（4）CMDR

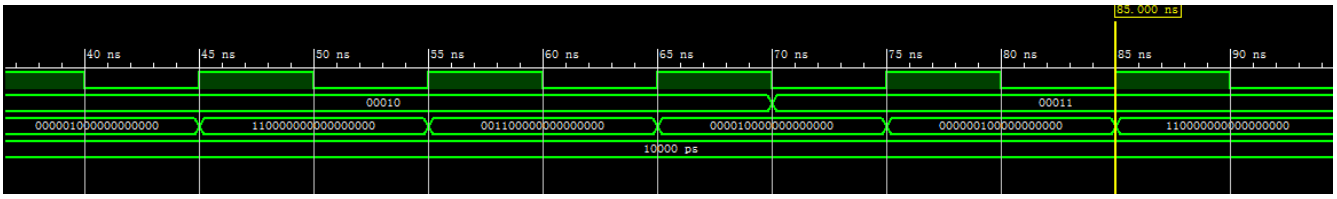


这里只载入了一次数据，在 25ns 处，第二行输入数据变为新值，恰逢时钟上升沿，输出改变。

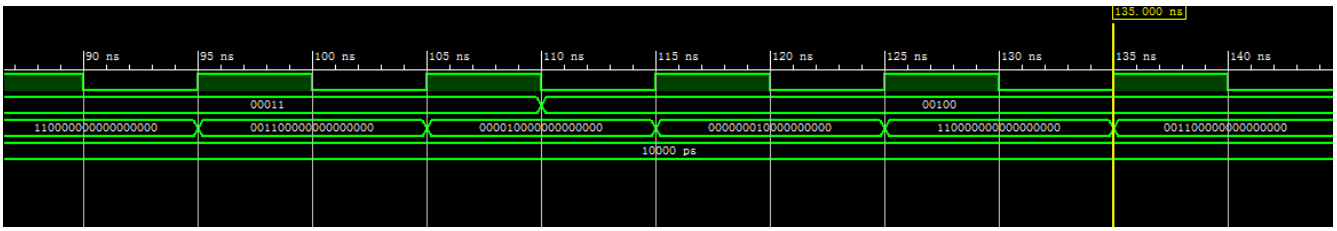
（5）总的 CU



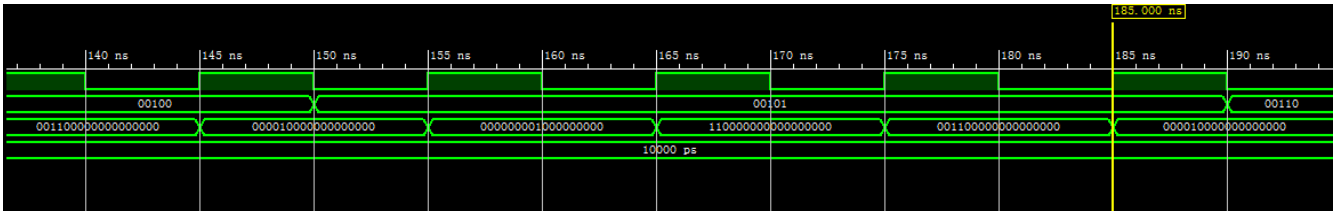
00H　　01H　　02H(跳转)　　04H



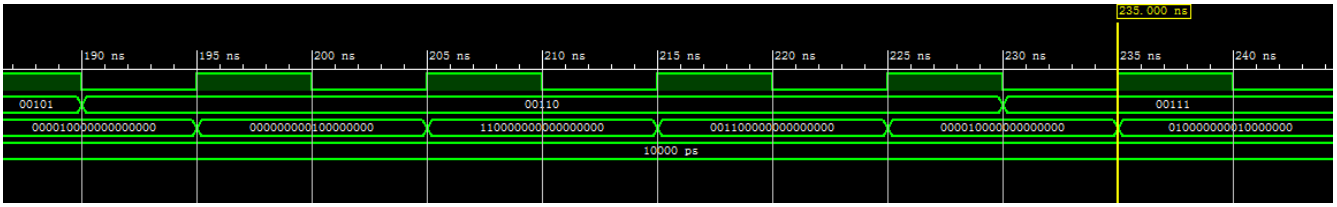
(返回 00H)　　00H　　01H　　02H　　06H　　00H



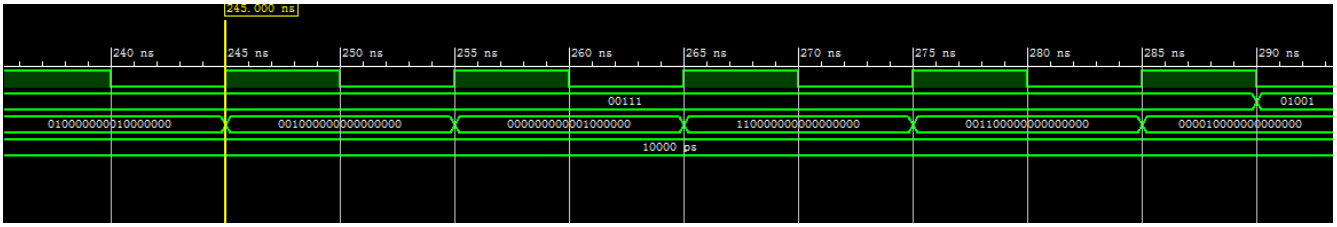
01H　　02H　　08H　　00H　　01H



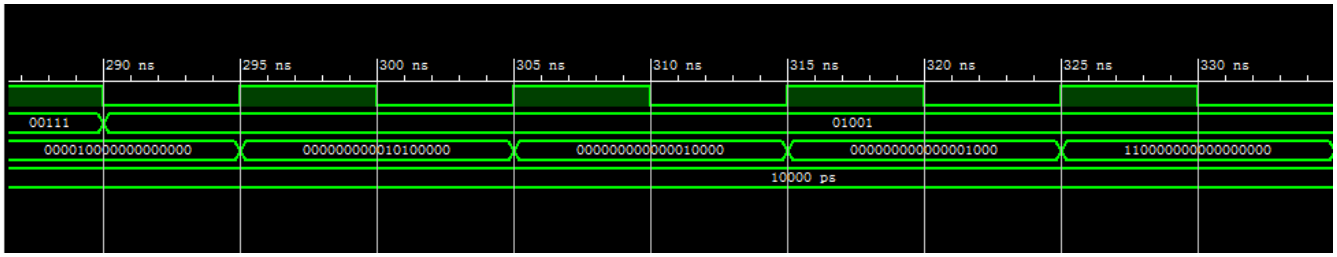
02H　　0AH　　00H　　01H　　02H



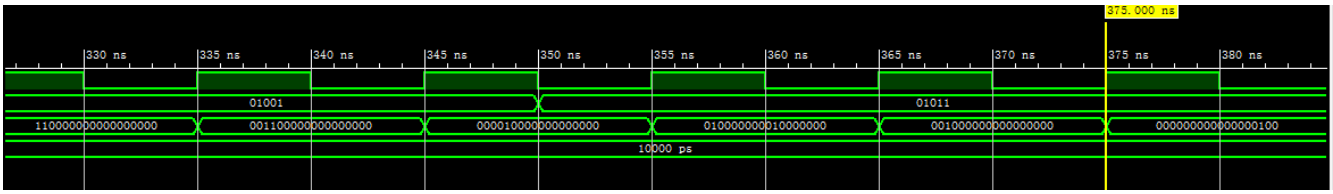
0CH　　00H　　01H　　02H　　0EH



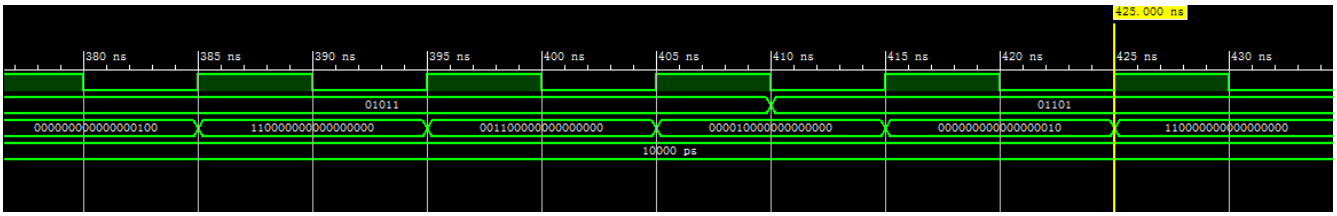
0FH　　10H　　00H　　01H



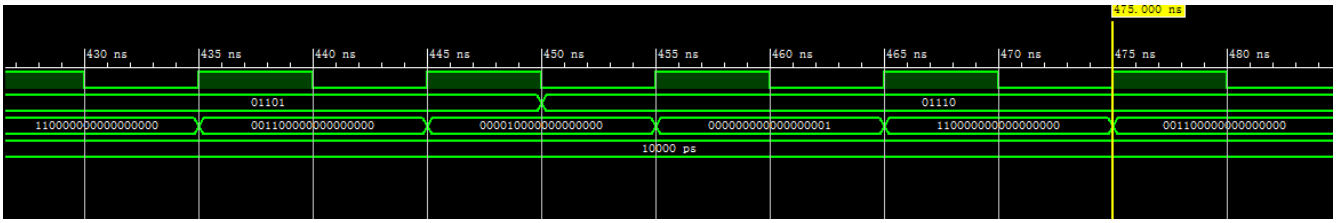
02H 12H 13H 14H 00H



01H 02H 16H 17H 18H



00H 01H 02H 1AH 00H



01H 02H ICH 00H