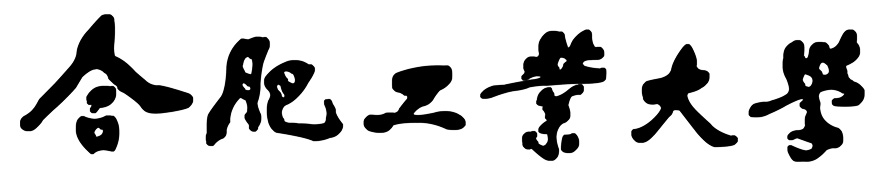
****

**《计算机组成原理》**

**实验报告**

**学生姓名**

**学 号**

**专业班级**

**指导教师 安鑫**

**院系名称 计算机与信息系**

**2020 年 1 月 4 日**

目 录

[0. 说明 1](#_Toc29068327)

[1. 实验目的 1](#_Toc29068328)

[2. 实验环境 1](#_Toc29068329)

[2.1.硬件环境 1](#_Toc29068330)

[2.2.软件环境 2](#_Toc29068331)

[3. 实验内容 2](#_Toc29068332)

[4. 实验原理 3](#_Toc29068333)

[4.1.根据功能和格式完成CPU的数据通路设计 3](#_Toc29068334)

[4.2.根据指令功能、数据通路完成控制单元的设计 3](#_Toc29068335)

[5. 实验步骤 5](#_Toc29068336)

[5.1.CPU各模块Verilog实现 5](#_Toc29068337)

[5.2.CPU顶层文件封装实现 11](#_Toc29068338)

[5.3.CPU模拟仿真 12](#_Toc29068339)

[6. 实验结果与分析 22](#_Toc29068340)

[7. 心得体会 24](#_Toc29068341)

# 说明

老师说过：“不申优不用贴代码”，我想反正自己都参考别人的、自己调试和自己写給弄出来了，所以就贴了代码，不过并未以附录的形式将所有源代码給出，而是只贴出了实验指导书上要求的部分。每段代码我都**注明**是自己写的还是参考了别人的。

另外，因为自己写的模块很多功能有遗缺，因此有些也借鉴了别人的代码。**实验分析**完全是自己总结的，可能存在错误的地方。

# 实验目的

通过设计并实现支持10条指令的CPU，进一步理解和掌握CPU设计的基本原理和过程。

# 实验环境

## 硬件环境

1. 平台：Dell G3 3579。
2. 配置：
3. 处理器：Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz；
4. 已安装的内存(RAM)：16.0 GB (15.9 GB可用)。

## 软件环境

1. 操作系统：Windows 10 家庭中文版；
2. 系统类型：64位操作系统，基于x64的处理器。
3. 开发工具：ModelSim SE-64 2019.2。

# 实验内容

设计和实现一个支持如下十条指令的单周期CPU。在本实验中寻址方式均采用寄存器寻址，对于双目运算操作可表示为

OP r1, r2, r3

单目运算操作可表示为

OP r1

具体格式约定如下：

⮚ 采用寄存器寻址，r1，r2，r3为寄存器编号，r1和r2存放两个源操作数，r3为目标寄存器，对于双目运算其功能为[r1] op [r2]->r3，而对于单目运算其功能为op [r1]->r3；

⮚ 指令字长16位，操作码和地址码字段分配如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 15 12 | 11 8 | 7 4 | 3 0 |
| OpCode | r1 | r2 | r3 |

⮚ **非访存指令**

◆ 清除累加器指令 CLA

◆ 累加器取反指令 COM

◆ 算术右移一位指令 SHR

◆ 循环左移一位指令 CSL

◆ 停机指令 STP

⮚ **访存指令**

◆ 加法指令 ADD X： [X] + [ACC] –〉 ACC，X为存储器地址，直接寻址

◆ 存数指令 STA X，采用直接寻址方式

◆ 取数指令 LDA X，采用直接寻址

⮚ **转移类指令**

◆ 无条件转移指令 JMP imm： signExt(imm) -> PC

◆ 有条件转移（负则转）指令 BAN X: ACC 最高位为 1 则（ PC） + X -> PC,否则 PC不变

# 实验原理

单周期CPU是指所有指令均在一个时钟周期内完成的CPU。CPU由数据通路及其控制部件两部分构成，因而要完成一个支持若干条指令CPU的设计，需要依次完成以下两件事：

1. 根据指令功能和格式设计CPU的数据通路；
2. 根据指令功能和数据通路设计控制部件。

## 根据功能和格式完成CPU的数据通路设计

本实验需要设计的CPU需要支持第3节所述的10条指令，指令的含义可以参考第3节的有关介绍。因指令条数较多，分别給出每条指令的数据通路容易使人困惑，因此本节并未对其进行详细描述。

## 根据指令功能、数据通路完成控制单元的设计

控制单元的功能是为当前要执行的指令产生微操作命令从而完成该指令的执行。这里直接給出了整合控制单元后目标CPU的原理图，如图4.1所示，下面的实验都将围绕它展开。



图 4.１16位单周期CPU原理图



图 4.２ CPU工作过程

# 实验步骤

在第三部分通过对该CPU实现细节的分析、设计，并得到该CPU的原理图后，就可以依次实现各个模块，并进行仿真验证了。

## CPU各模块Verilog实现

在之前的实验中（按老师的要求这里并未給出先序实验的介绍），已经分别设计和实现了PC、指令存储器、寄存器组和ALU，这里只给出各个模块的功能描述及其接口定义，具体实现可以直接使用或者调整前面试验的实现代码。

1. **PC模块**

表 5.１ PC模块功能描述

|  |  |
| --- | --- |
| 输入 | 时钟信号clk、重置信号rst |
| 输出 | 指令地址pc（8位） |
| 功能 | 每个时钟上升沿PC的值自动加1，并输出 |

Verilog关键代码（**完全自己写的**）：

**module** MyPC**(**

**input** clk**,**

**input** rst**,**

//input wire [7:0] initAddr,

**output** **[**7**:**0**]** pcAddr

**);**

**reg** **[**7**:**0**]** tmpPC **=** 8'b0**;**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)**

tmpPC **=** 8'b0**;**//8'b01011010;

//else if(tmpPC == 8'b0)

//tmpPC = initAddr;

**else**

tmpPC **=** tmpPC **+** 1'b1**;**

**end**

**assign** pcAddr **=** tmpPC**;**

**endmodule**

1. **指令存储器模块**

表 5.２指令存储器模块功能描述

|  |  |
| --- | --- |
| 输入 | 8位指令地址Addr |
| 输出 | 16位指令Ins |
| 功能 | 存放待执行的指令（初始化），并根据地址输出指令 |

Verilog关键代码（**有借鉴别人的**）：

**module** ins**(**

**output** **reg[**15**:**0**]** outins**,**

**input** **wire[**7**:**0**]** adder**,**

**output** **reg[**5**:**0**]** r1**,**

**output** **reg[**7**:**0**]** r2**,**

**input** **wire** clk

**);**

**reg** **[**15**:**0**]** storage **[**0**:**255**];**

**reg** **[**15**:**0**]** temp**;**

**initial** **begin**

storage **[**8'h00**]** **=** 16'h0000**;** //start

storage **[**8'h01**]** **=** 16'h0100**;** //cla

storage **[**8'h02**]** **=** 16'h0200**;** //com

storage **[**8'h03**]** **=** 16'h0300**;** //shr

storage **[**8'h04**]** **=** 16'h0400**;** //csl

storage **[**8'h05**]** **=** 16'h0500**;** //add x

storage **[**8'h06**]** **=** 16'h0600**;** //sta x

storage **[**8'h07**]** **=** 16'h0701**;** //lda x

storage **[**8'h08**]** **=** 16'h0803**;** //jmp

storage **[**8'h09**]** **=** 16'h0902**;** //ban

storage **[**8'h0a**]** **=** 16'hffff**;** //stop

**end**

**always** **@(**adder**)begin**

temp **=** storage**[**adder**];**

r1 **=** **{**storage**[**adder**][**5**:**0**]};**

r2 **=** **{**storage**[**adder**][**7**:**0**]};**

outins **=** temp**;**

**end**

**endmodule**

1. **寄存器堆**

表 5.３寄存器堆模块功能描述

|  |  |
| --- | --- |
| 输入 | 时钟信号clk、读写控制线wr\_en、读寄存器编号read\_reg1和 read\_reg2、写寄存器编号write\_reg、写入数据write\_data |
| 输出 | 对应两个读寄存器编号的寄存器值reg1和reg2 |
| 功能 | 根据读寄存器编号给出对应寄存器的值；在写允许情况下，把写入端的数据在clk下降沿写到写寄存器编号对应的寄存器 |

Verilog关键代码（**完全自己写的**）：

**module** registerBank**(**

**input** **[**2**:**0**]** addrA**,**addrB**,**inAddr**,**

**output** **wire[**15**:**0**]** outA**,**outB**,**

**input** **wire[**15**:**0**]** inData**,**

**input** clk**,**WE

**);**

**reg[**15**:**0**]** regs**[**0**:**7**];**

**assign** outA **=** regs**[**addrA**];**

**assign** outB **=** regs**[**addrB**];**

**always@(negedge** clk**)begin**

**if(** WE **==** 1 **)**

regs**[**inAddr**]** **=** inData**;**

**end**

**endmodule**

1. **ALU**

表 5.４ALU 模块功能描述

|  |  |
| --- | --- |
| 输入 | 操作数in1和in2、操作选择信号alu\_op |
| 输出 | ALU运算结果Z |
| 功能 | 根据操作选择信号计算in1和in2的运算结果Z |

Verilog关键代码（**完全自己写的**）：

**module** ALU**(**

**input** **wire** **[**15**:**0**]** x**,**

**input** **wire** **[**15**:**0**]** y**,**

**input** **wire** **[**3**:**0**]** select**,**

**output** **[**15**:**0**]** ans**,**//ans = answer

//status[0:3] //0: zero, 1: carry, 2: negative, 3: overflow

//output reg [3:0] status

**output** zero**,**

**output** carry**,**

**output** negative**,**

**output** overflow

**);**

/\*

???

???

???

???

???

???

???

???

\*/

**wire** **signed** **[**15**:**0**]** tmpX **=** x**,** tmpY **=** y**;**

**reg** **[**16**:**0**]** result **=** 17'b0**;**//??????????16??17???????

**always@\*** **begin**

**case(**select**)**

3'b000**:**

result **=** x **+** y**;**

3'b001**:**

result **=** tmpX **+** tmpY**;**

3'b010**:**

result **=** x **-** y**;**

3'b011**:**

result **=** tmpX **-** tmpY**;**

3'b100**:**

result **=** x **&** y**;**

3'b101**:**

result **=** x **|** y**;**

3'b110**:**

result **=** x **>>** y**;**

3'b111**:**

result **=** x **<<** y**;**

**default:**

result **=** 16'b1111111111111111**;**

**endcase**

**end**

**assign** ans **=** result**[**15**:**0**];**

**assign** zero **=** result **==** 16'b0 **?** 0 **:** 1**;**

**assign** negative **=** result**[**15**];**

**assign** overflow **=** result**[**16**];**

**endmodule**

1. **控制单元**

表 5.５控制单元模块功能描述

|  |  |
| --- | --- |
| 输入 | 指令（操作码） |
| 输出 | 寄存器堆的读写控制线wr\_en、ALU的操作选择信号alu\_op |
| 功能 | 根据当前指令功能对wr\_en和alu\_op赋值 |

Verilog关键代码（**有借鉴别人的**）：

**module** cu**(**

**input** **wire[**7**:**0**]** accout**,**

**input** **wire** **[**15**:**0**]** outins**,**

**output** **reg** stop**,**

**output** **reg** **[**1**:**0**]**accop**,**

**output** **reg** ena**,**

**output** **reg** **[**2**:**0**]** aluop**,**

**output** **reg** enable**,**

**output** **reg** pcJMP**,**banEBL**,**ban

**);**

**always** **@\*** **begin**

**case(**outins**[**15**:**8**])**

8'h00**:**

**begin**

stop **=** 0**;**

**end**

8'hff**:**

**begin**

stop **=** 1**;**

**end**

8'h01**:**

**begin**

ena **=** 1**;**

accop **=** 2'b00**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h02**:**

**begin**

ena **=** 1**;**

accop **=** 2'b01**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h03**:**

**begin**

ena **=** 1**;**

accop **=** 2'b10**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h04**:**

**begin**

ena **=** 1**;**

accop **=** 2'b11**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h05**:**

**begin**

ena **=** 1**;**

enable **=** 0**;**

aluop **=** 3'b000**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h06**:**

**begin**

ena **=** 1**;**

enable **=** 1**;**

banEBL **=** 0**;**

pcJMP**=**0**;**

**end**

8'h07**:**

**begin**

ena **=** 1**;**

enable **=** 0**;**

banEBL **=** 0**;**

pcJMP **=** 0**;**

**end**

8'h08**:**

**begin**

pcJMP **=** 1 **;**

**end**

8'h09**:**

**begin**

banEBL **=** 1 **;**

**if(**accout**[**7**]** **==** 1**)**

**begin**

ban **=** 1 **;**

**end**

**end**

**endcase**

**end**

**endmodule**

## CPU顶层文件封装实现

通过根据图4.1将以上定义的模块进行连接、封装就得到了目标CPU，该CPU的输入为系统时钟信号clk和重置信号reset。

Verilog关键代码（有借鉴别人的）：

**module** cpu**(**

**input** clk**,**

**input** rst**,**

**output** **wire** **[**7**:**0**]** pc

**);**

**wire** **[**2**:**0**]** RA**,** RB**,** RW**;**

**wire** **[**15**:**0**]** busW**;**

**wire** WE**,** RD**;**

**wire** **[**15**:**0**]** busA**,** busB**;**

**wire** **[**15**:**0**]** accout**;**

**wire** **[**2**:**0**]** aluop**;**

**wire** stop**,** pcJMP**,** banEBL**,** ban**,** ena**,** enable**;**

**wire** **[**15**:**0**]** outins**;**

**wire** **[**1**:**0**]** accop**;**

**wire** **[**5**:**0**]** r1**;**

**wire** **[**7**:**0**]** r2**;**

**wire** **[**15**:**0**]** dataout**,** out1**,** out2**;**

alu malu**(.**outalu**(**out1**),.**a**(**accout**),.**b**(**out2**),.**select**(**aluop**));**

pc mpc**(.**stop**(**stop**),.**clk**(**clk**),.**rst**(**rst**),.**pcJMP**(**pcJMP**),.**banEBL**(**banEBL**),.**ban**(**ban**),.**data\_in**(**r2**),.**pc**(**pc**));**

ins mins**(.**outins**(**outins**),.**adder**(**pc**),.**r1**(**r1**),.**r2**(**r2**),.**clk**(**clk**));**

cu mcu**(.**accout**(**accout**),.**outins**(**outins**),.**stop**(**stop**),.**accop**(**accop**),.**ena**(**ena**),.**aluop**(**aluop**),.**enable**(**enable**),.**pcJMP**(**pcJMP**),.**banEBL**(**banEBL**),.**ban**(**ban**));**

acc myacc**(.**ena**(**ena**),.**accop**(**accop**),.**accin**(**out1**),.**acc**(**accout**));**

register mregister**(**RA**,** RB**,** RW**,** busW**,** clk**,** WE**,** busA**,** busB**,** RD**);**

Datastorage mDatastorage**(.**address**(**r1**),.**enable**(**enable**),.**clk**(**clk**),.**dataout**(**dataout**),.**datain**(**accout**));**

**endmodule**

## CPU模拟仿真

为了仿真验证所实现的CPU，需要定义测试文件并在测试文件中对指令存储器和寄存器堆中的相应寄存器的值进行初始化，并通过仿真波形图查看是否指令得到了正确执行。

1. TestBench关键代码（**改的**）

**module** MyCPU\_tb**;**

**parameter** CYCLE **=** 32**;**

**reg** clk**;**

**wire** **[**7**:**0**]** pc**;**

**reg** rst**;**

cpu MyCPU**(**clk**,** rst**,** pc**);**

**initial** clk **=** 0**;**

//always #20 clk = ~clk;

**always** **#(**CYCLE**/**2**)** clk **=** **~** clk**;**

**initial** **begin**

clk **=** 1**;**

rst **=** 0**;**

**#(**CYCLE**\***1**)**

clk **=** 1**;**

rst **=** 1**;**

**#(**CYCLE**\***2**)**

rst **=** 0**;**

**#(**CYCLE**\***3**)**

rst **=** 0**;**

**#(**CYCLE**\***4**)**

rst **=** 0**;**

**#(**CYCLE**\***5**)**

rst **=** 1**;**

**#(**CYCLE**\***6**)**

rst **=** 0**;**

**#(**CYCLE**\***7**)**

rst **=** 0**;**

**#(**CYCLE**\***8**)**

rst **=** 0**;**

**#(**CYCLE**\***9**)**

rst **=** 1**;**

**#(**CYCLE**\***9**)**

rst **=** 0**;**

**force** MyCPU**.**r1 **=** 7'h2**;**

**force** MyCPU**.**r2 **=** 8'h3**;**

**#(**CYCLE**\***10**)**

**force** MyCPU**.**outins **=** 16'h0600**;**//16'h0500

**force** MyCPU**.**banEBL **=** 1**;**

**#(**CYCLE**\***12**)**

**force** MyCPU**.**accout**[**7**]** **=** 1**;**

**force** MyCPU**.**r2 **=** 16'h2**;**

**#(**CYCLE**\***13**)**

//force MyCPU.accout[7] = 1;

**force** MyCPU**.**outins **=** 16'h0902**;**

//force MyCPU.outins = 16'hffff;//.outins(16'hffff);

**#(**CYCLE**\***14**)**

**force** MyCPU**.**outins **=** 16'hffff**;**

**#(**CYCLE**\***20**)**

$stop**;**

**end**

**endmodule**

1. ModelSim仿真结果

仿真结果主要以截图的形式展示，如图5.1至图5.26所示。

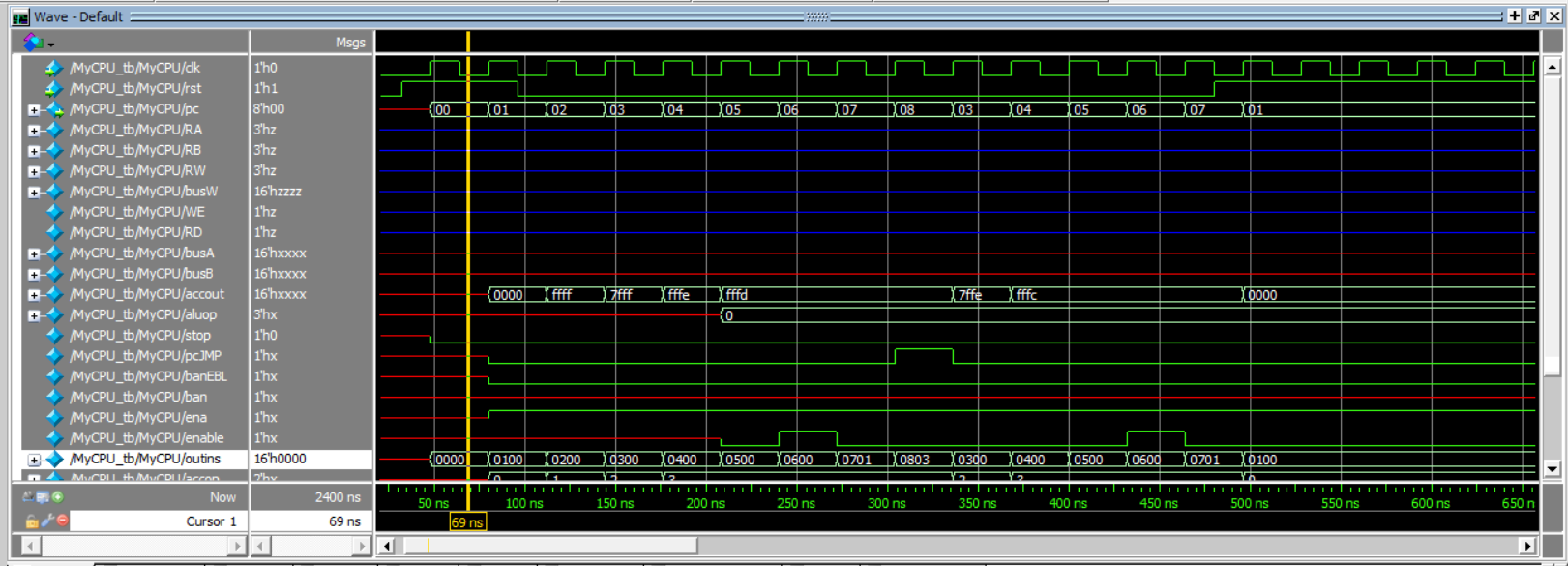


图 5.１ 执行开机指令START 1（“1”指的是参数的上半部分，下同）

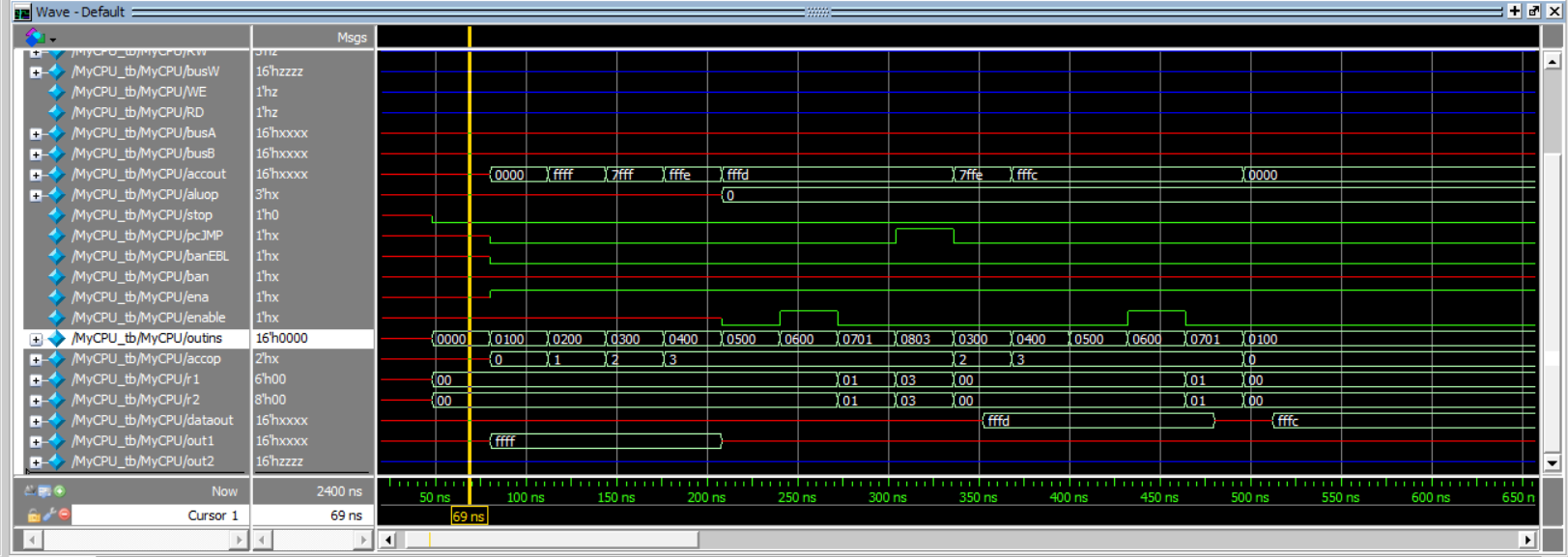


图 5.２ 执行开机指令START 2（“2”指的是参数的下半部分，下同）

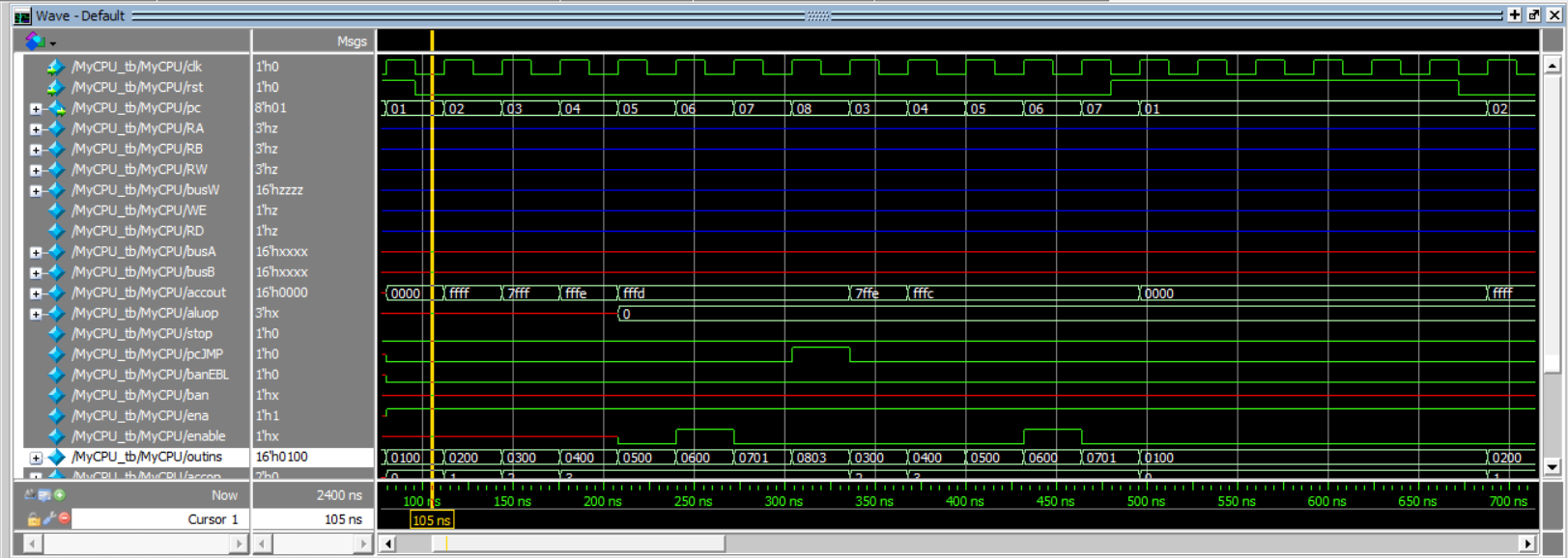


图 5.３ 执行清除累加器指令CLA 1

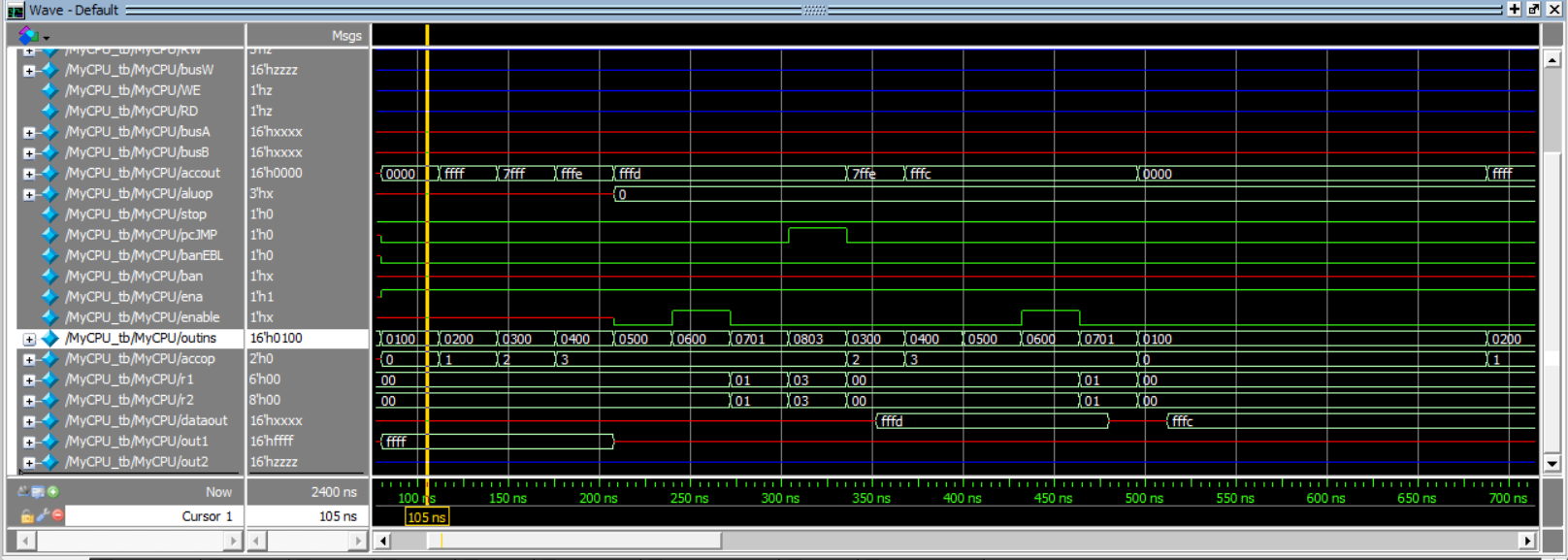


图 5.４ 执行清除累加器指令CLA 2

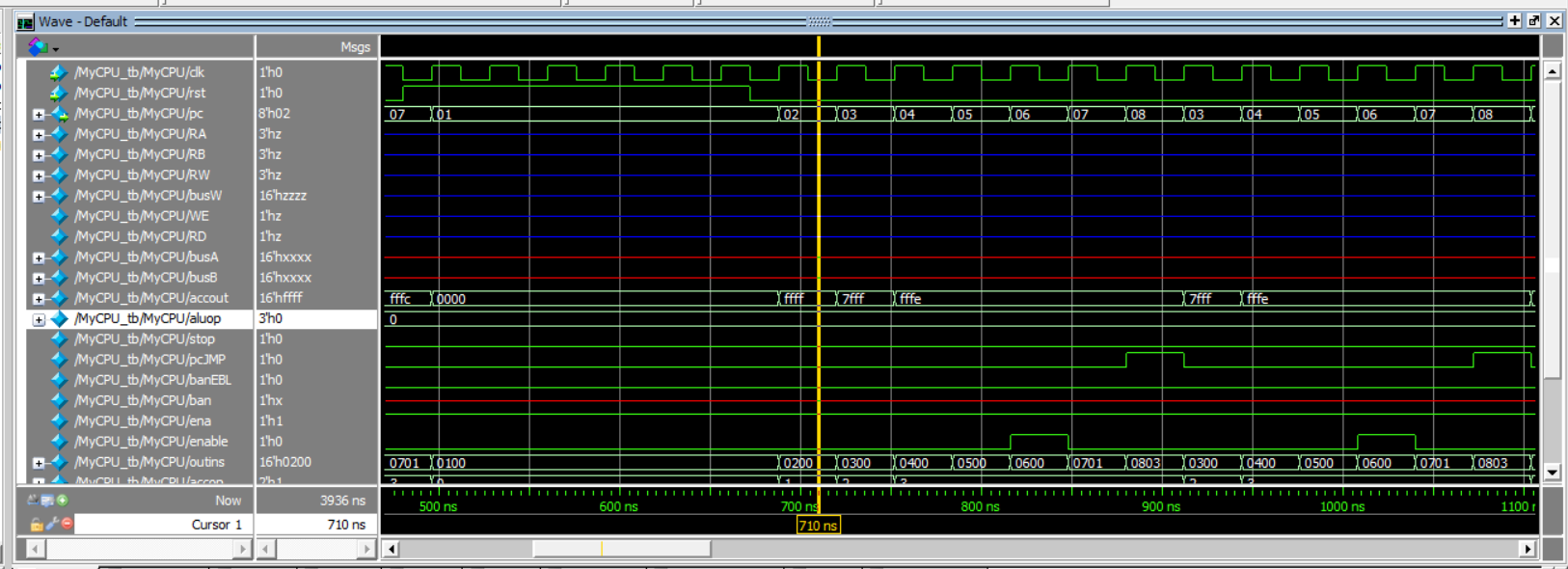


图 5.５ 执行累加器取反指令COM 1

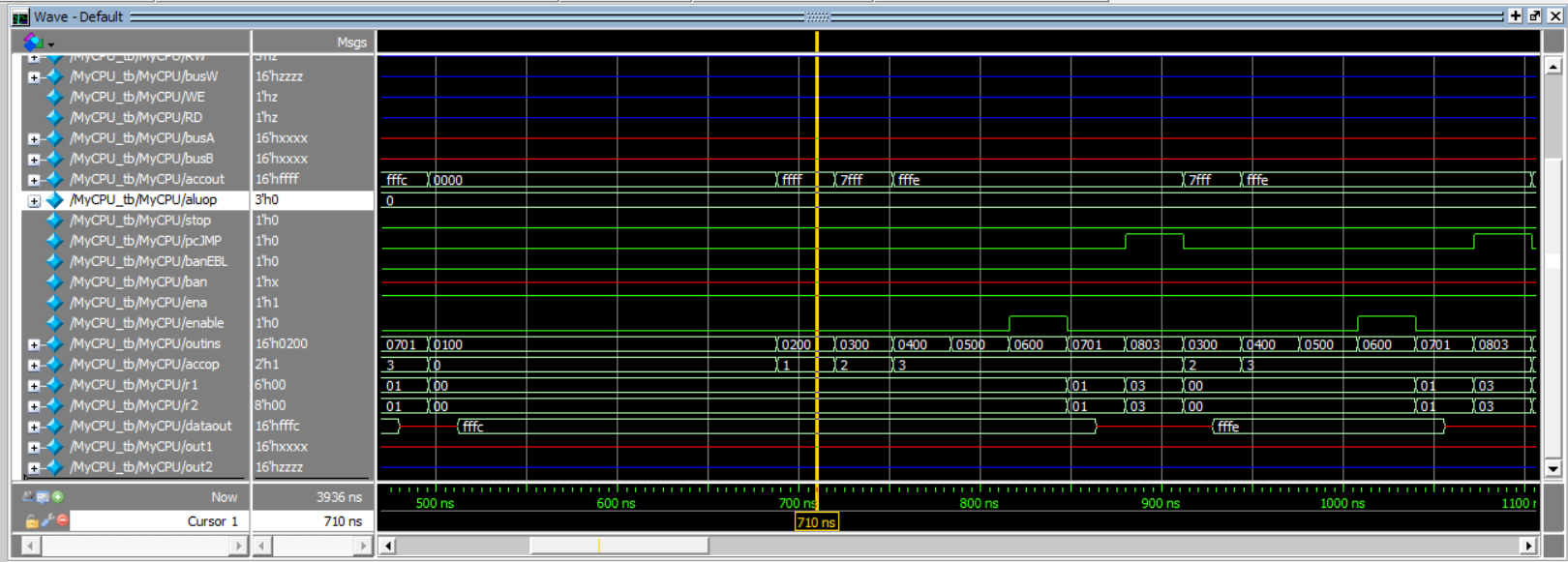


图 5.６ 执行累加器取反指令COM 2

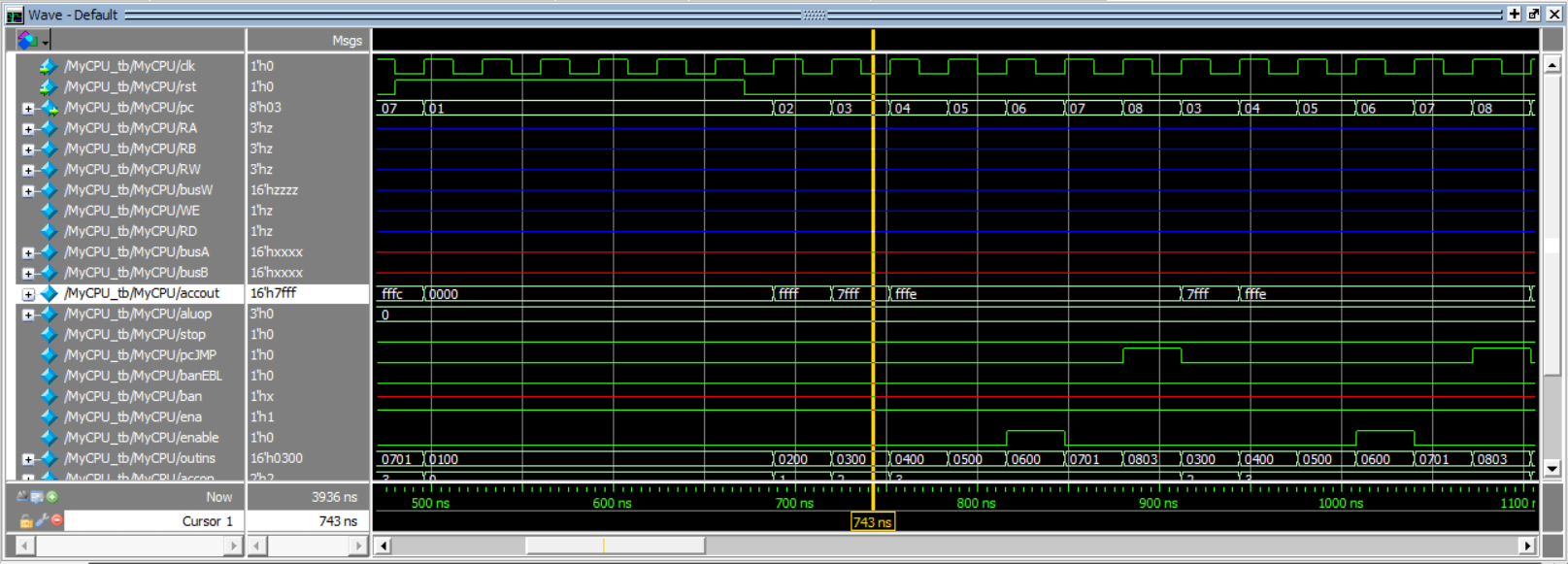


图 5.７ 执行算术右移一位指令SHR 1

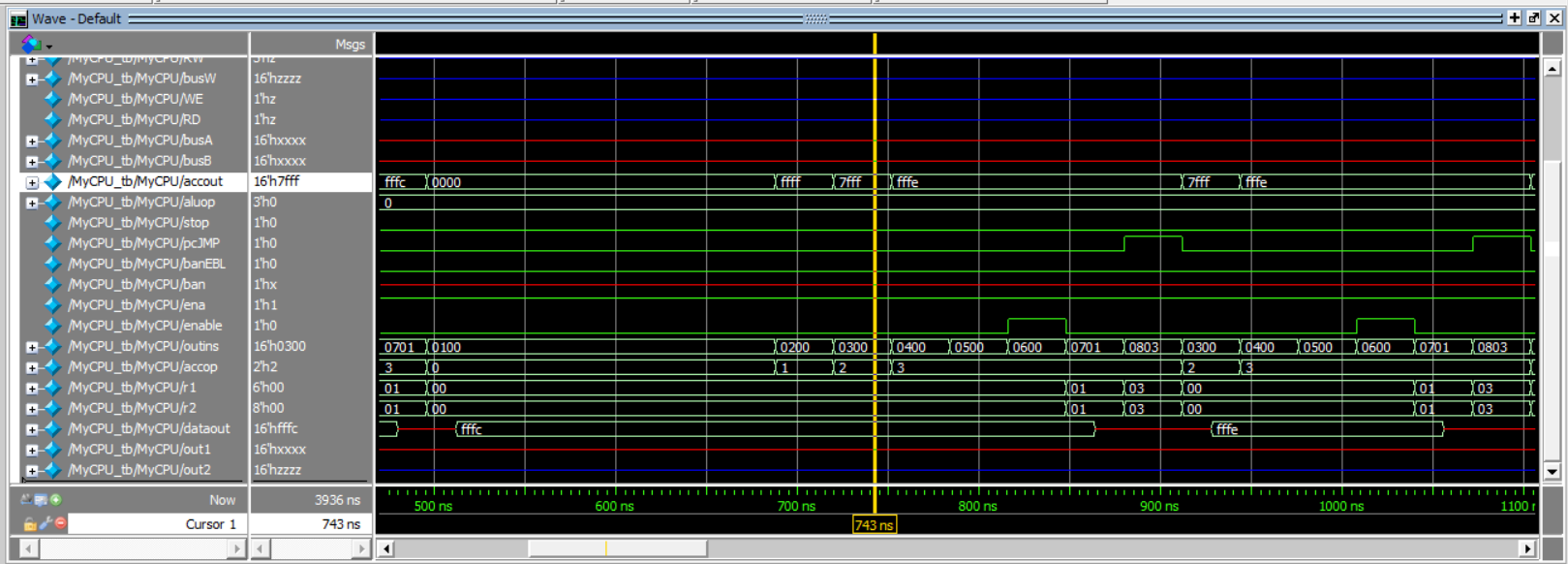


图 5.８ 执行算术右移一位指令SHR 2

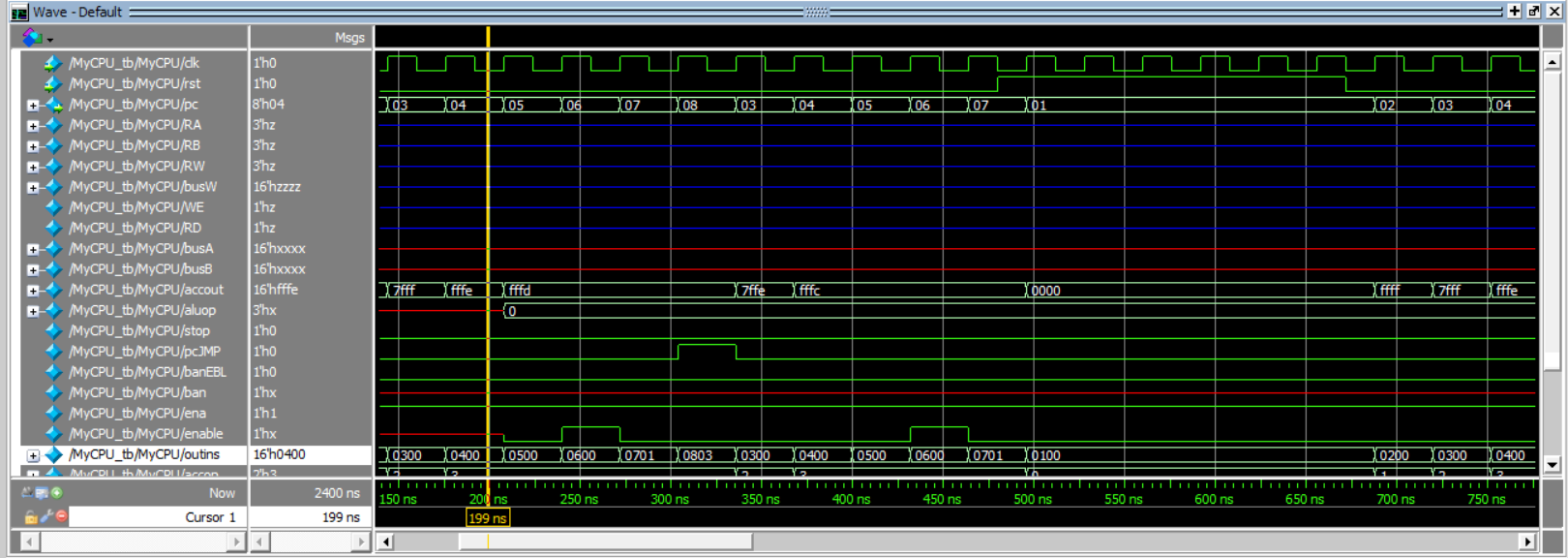


图 5.９ 执行循环左移一位指令CSL 1

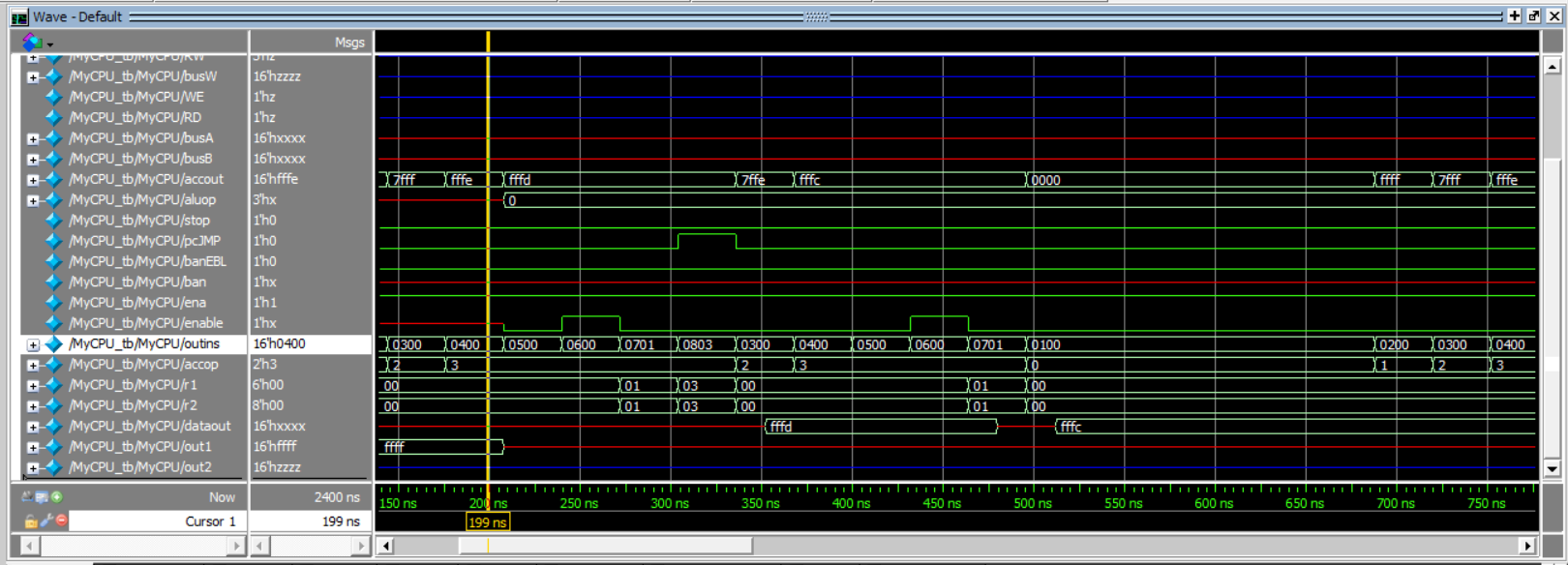


图 5.１０ 执行循环左移一位指令CSL 2

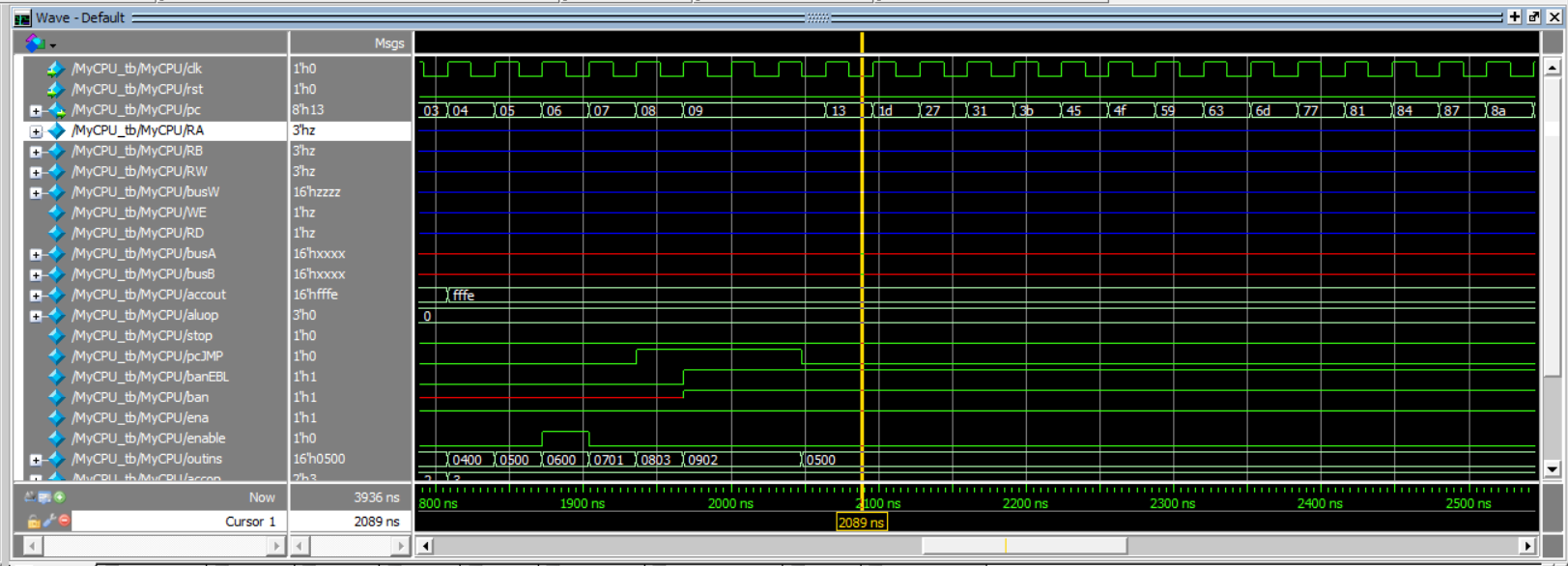


图 5.１１ 执行加法指令ADD 1

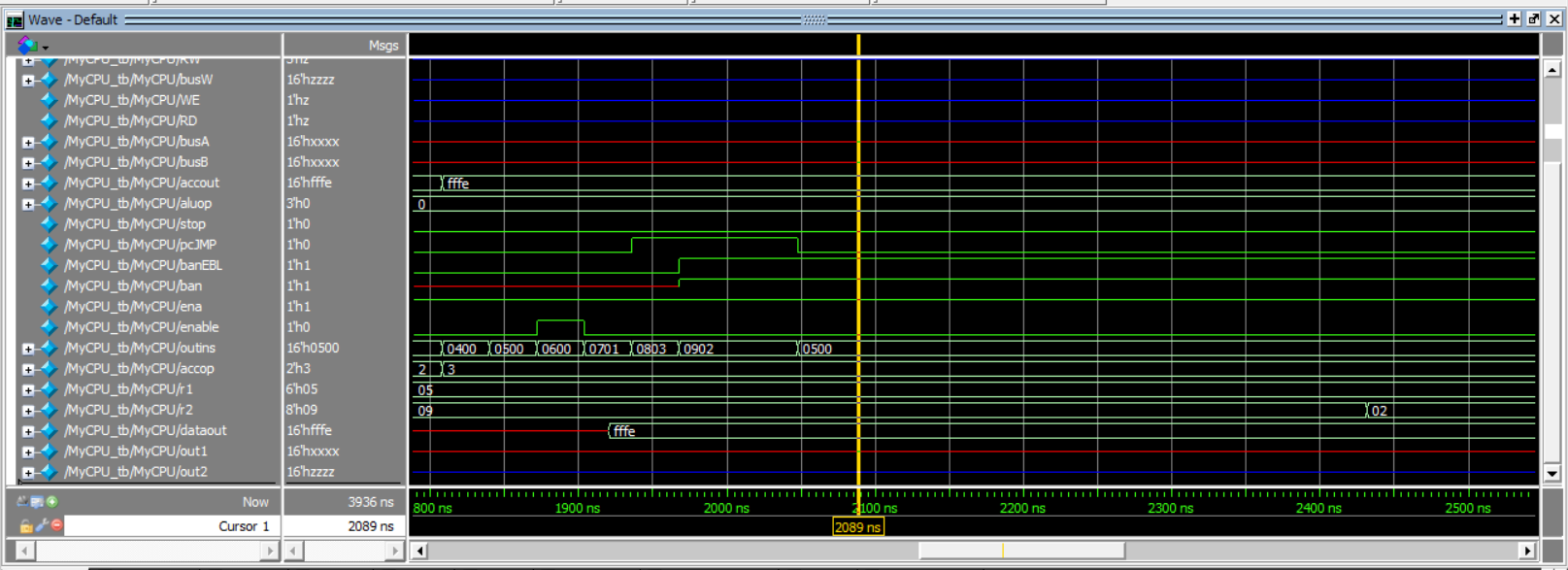


图 5.１２ 执行加法指令ADD 2

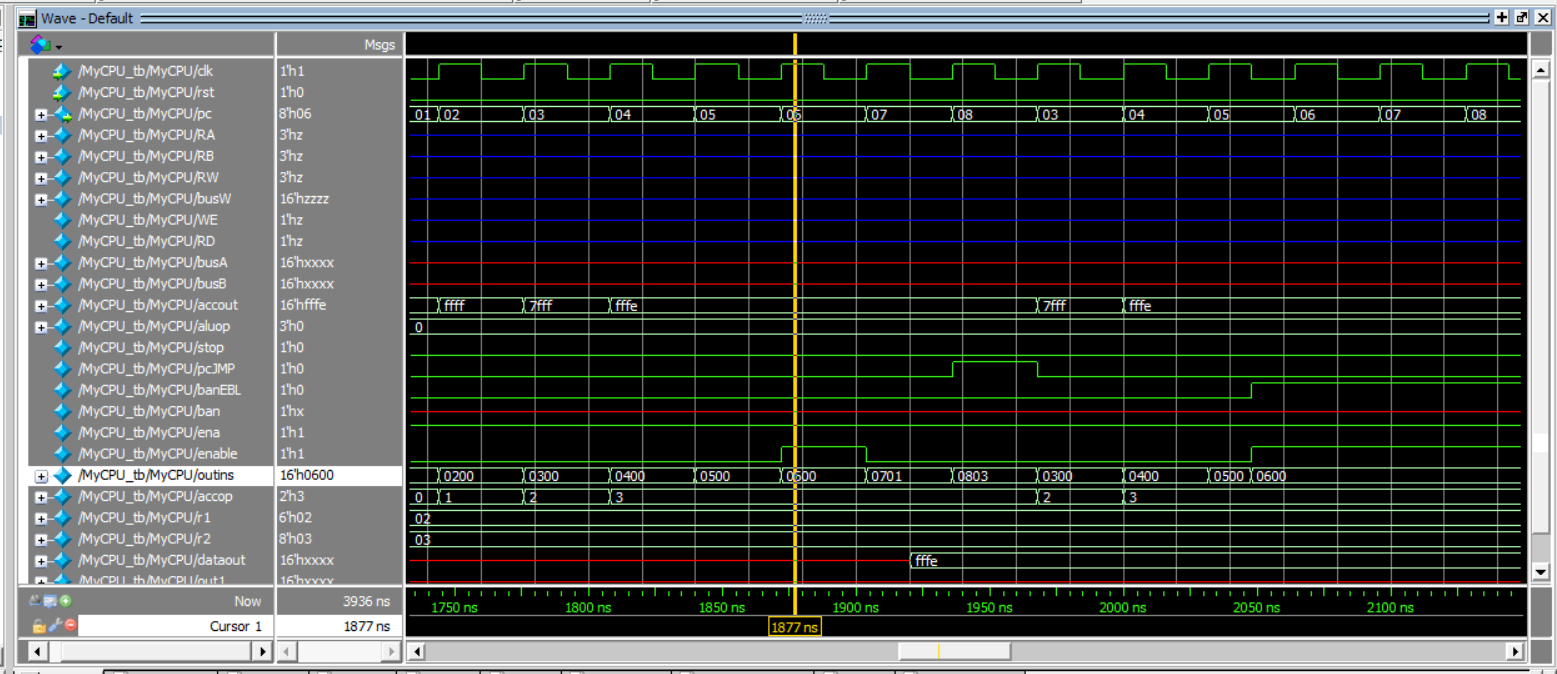


图 5.１３ 执行存数指令STA 1

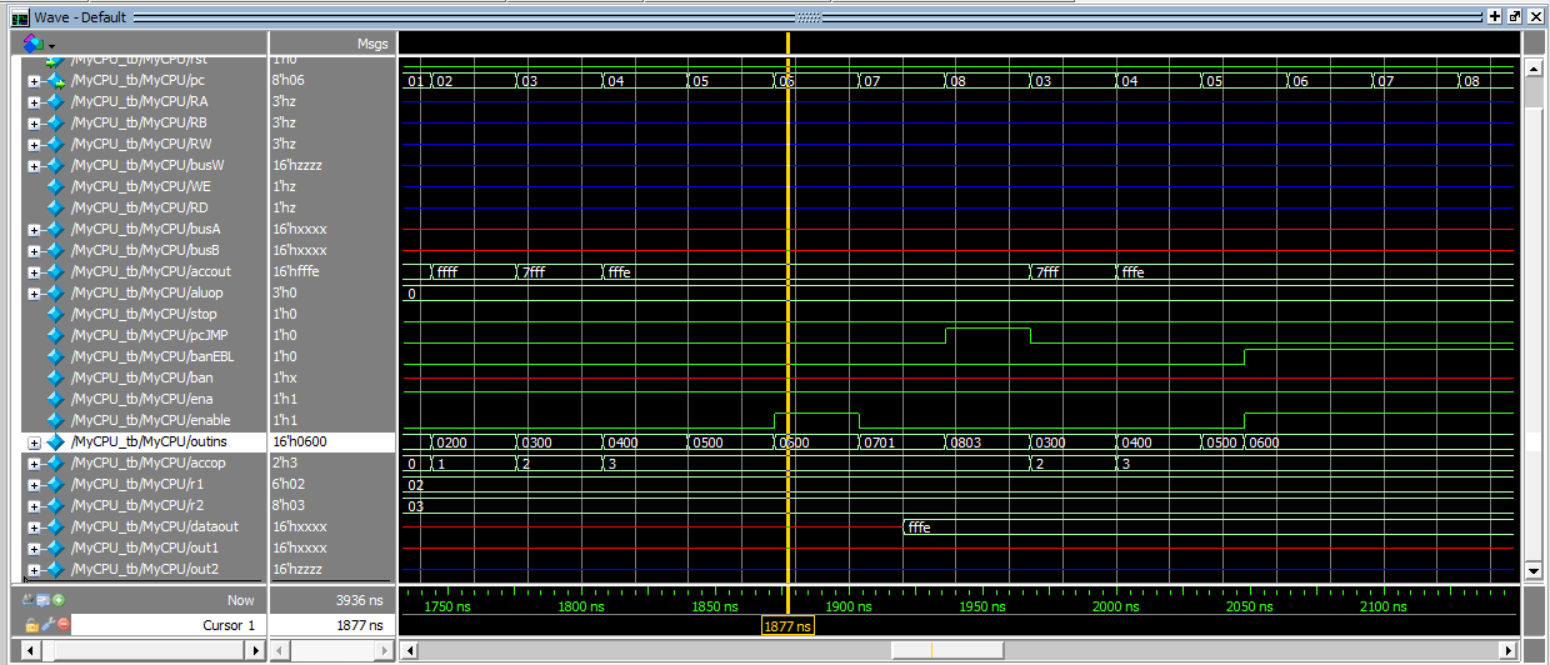


图 5.１４ 执行存数指令STA 2

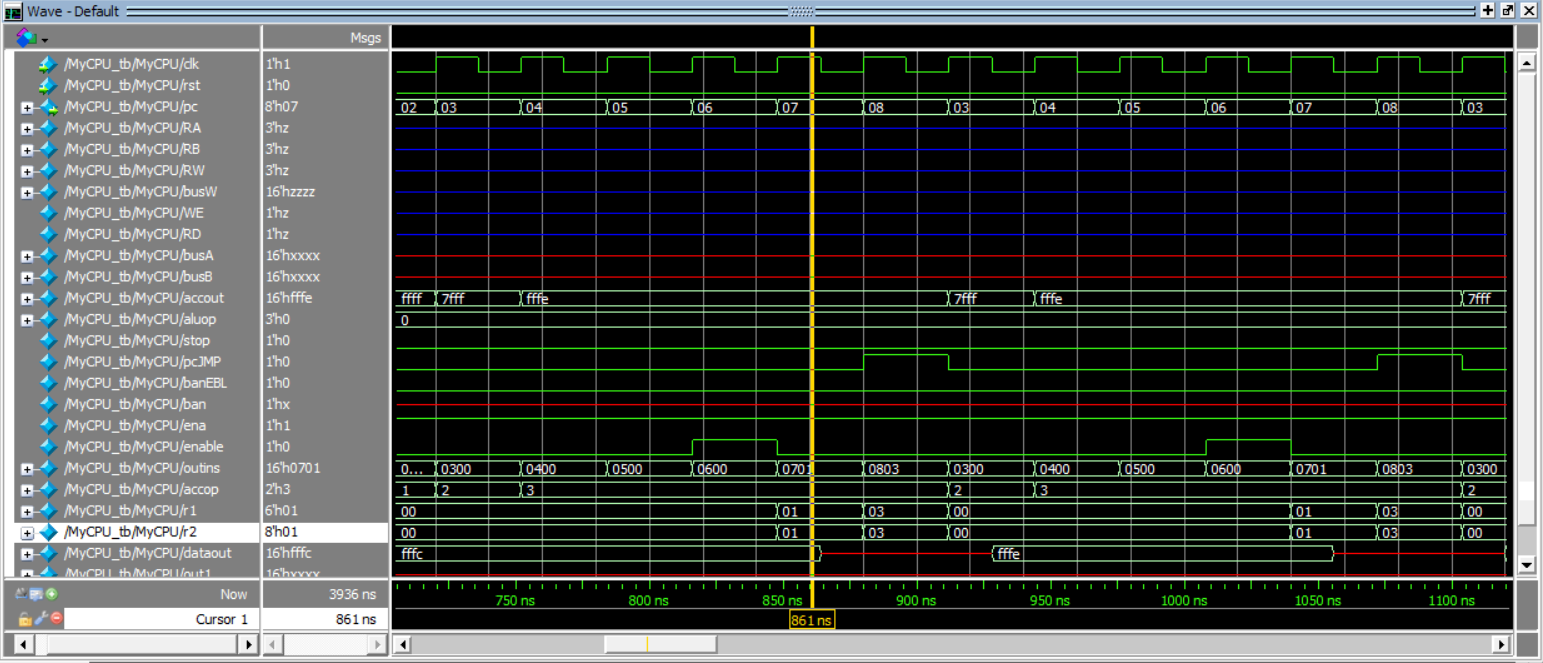


图 5.１５ 执行取数指令LDA 1

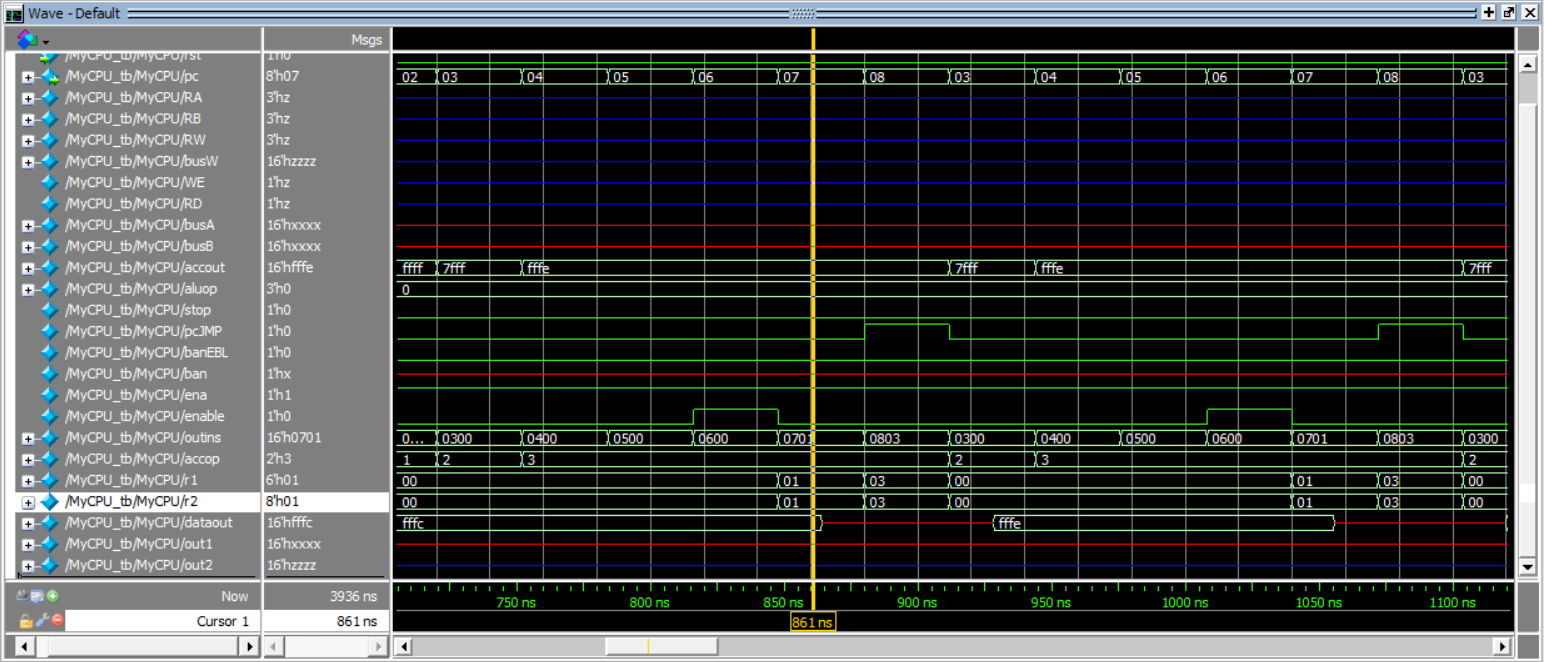


图 5.１６ 执行取数指令LDA 2

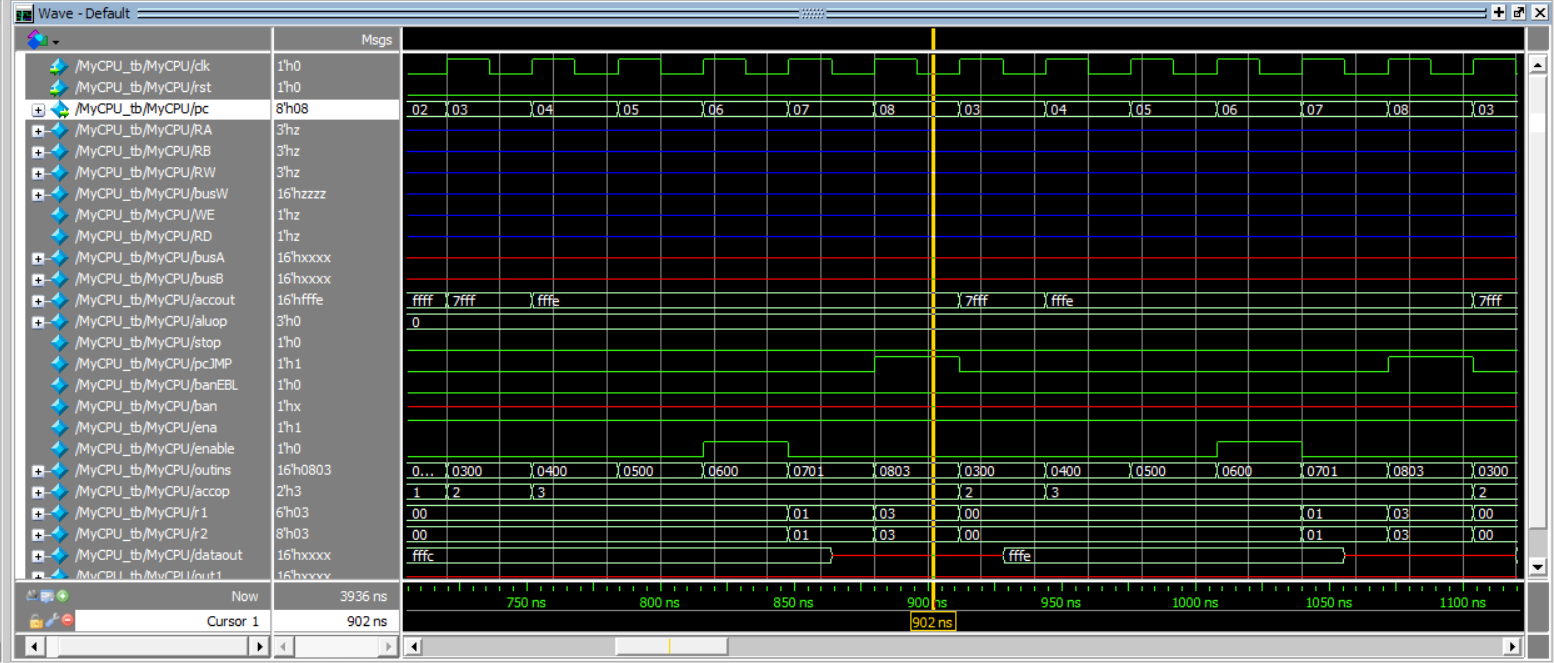


图 5.１７ 执行无条件转移指令JMP前 1

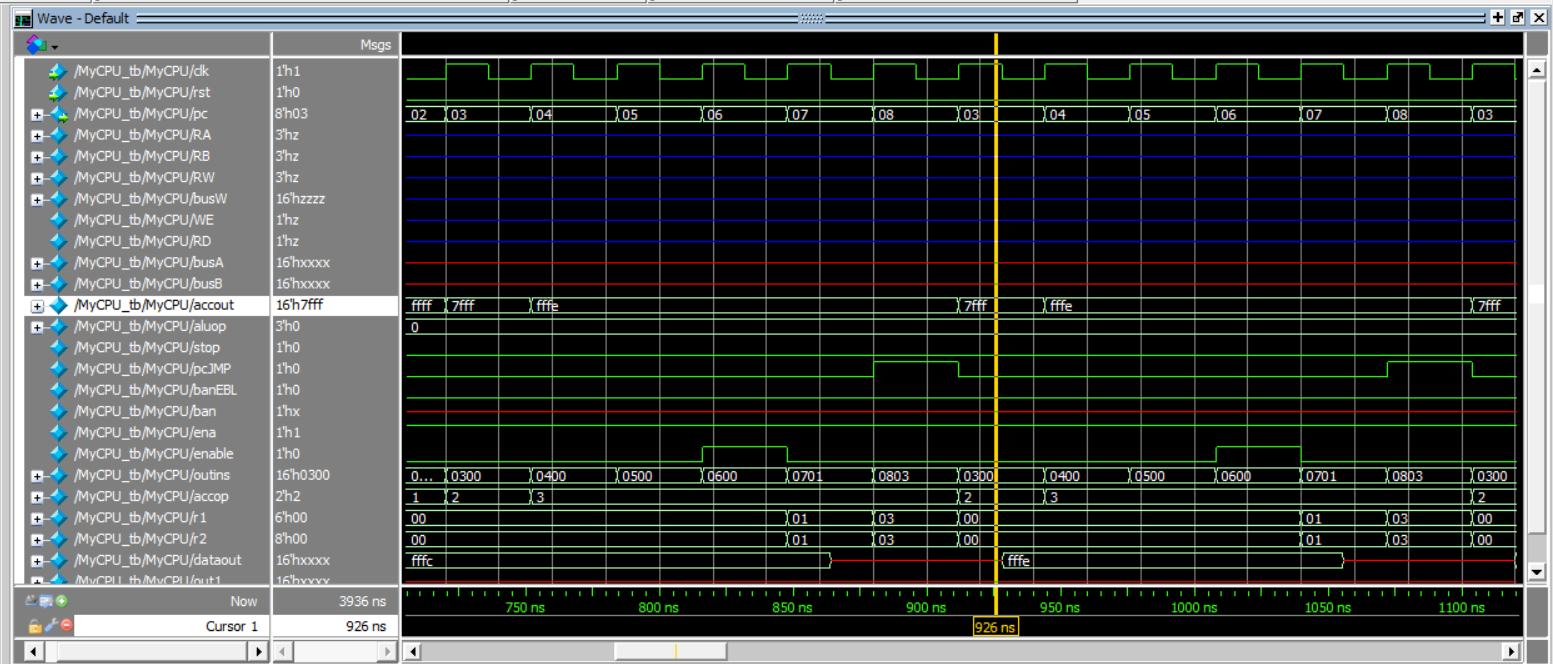


图 5.１８ 执行无条件转移指令JMP后 1

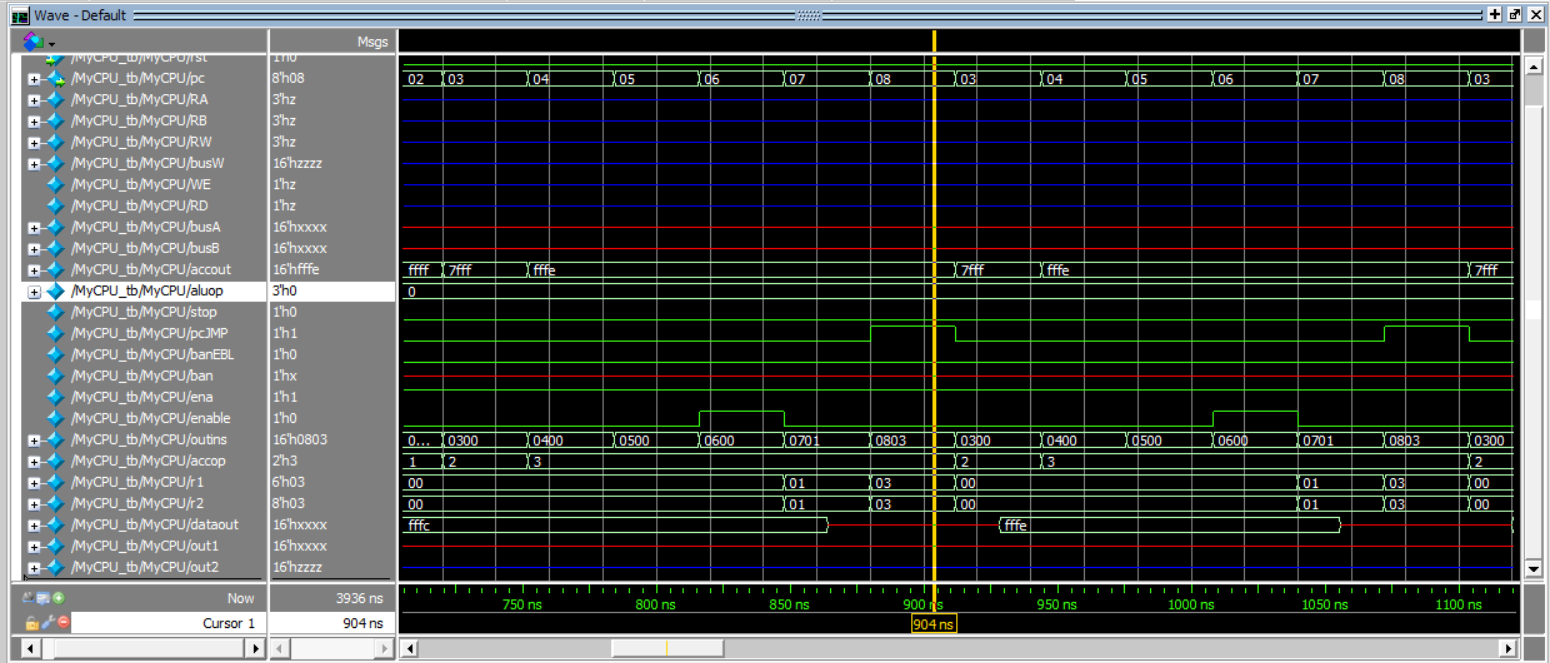


图 5.１９ 执行无条件转移指令JMP前 2

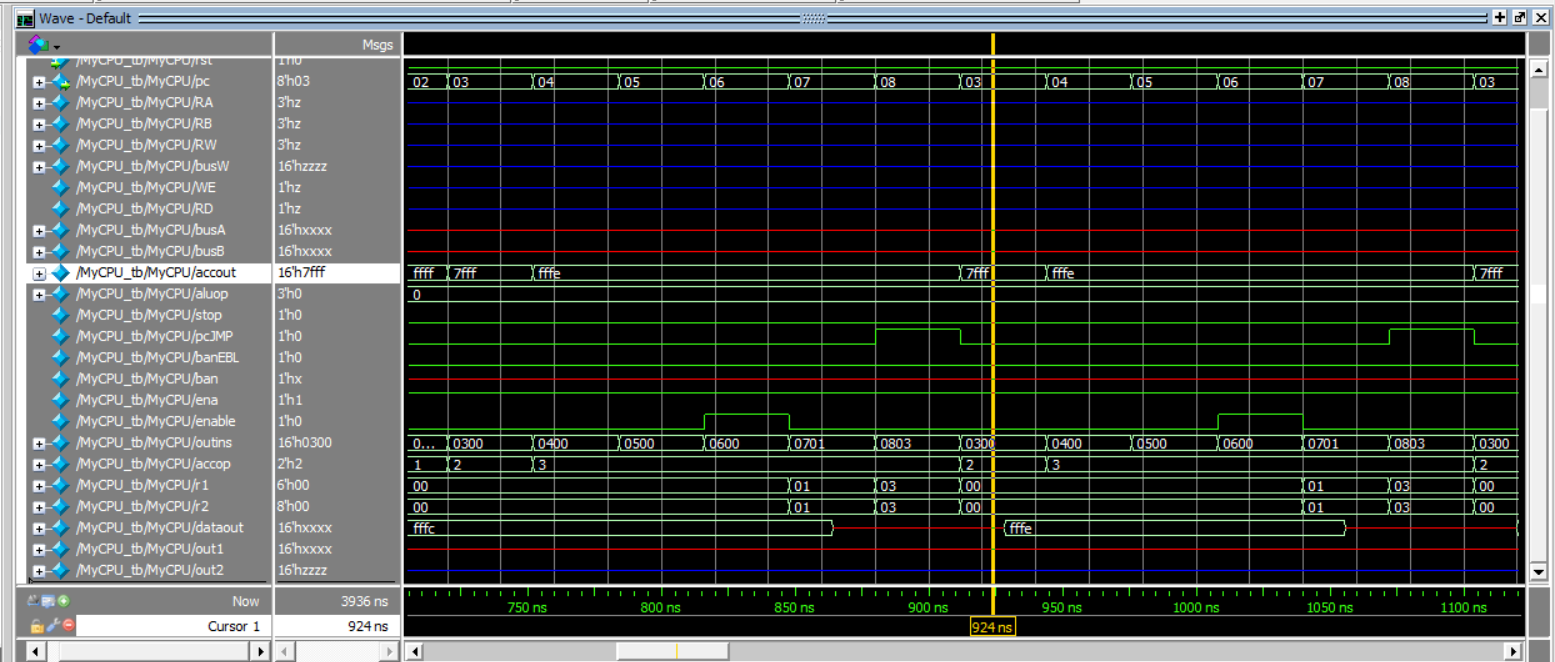


图 5.２０ 执行无条件转移指令JMP后 2

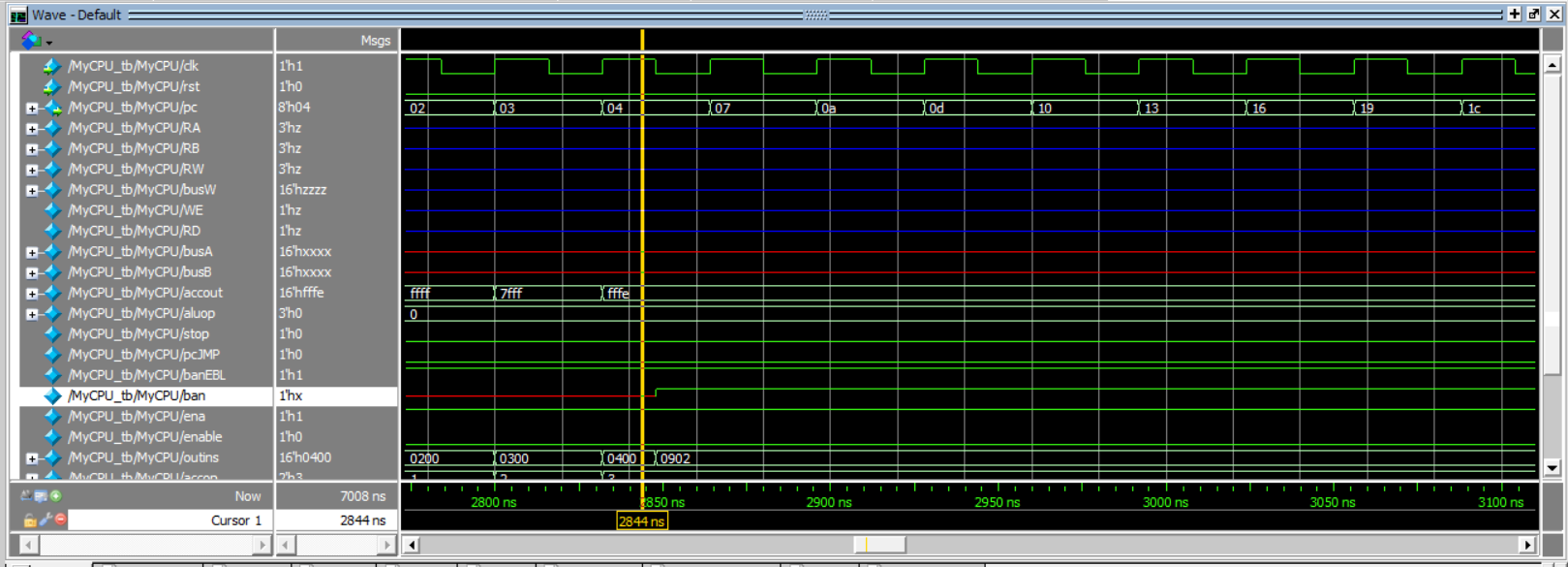


图 5.２１ 执行有条件转移（负则转）指令BAN前 1

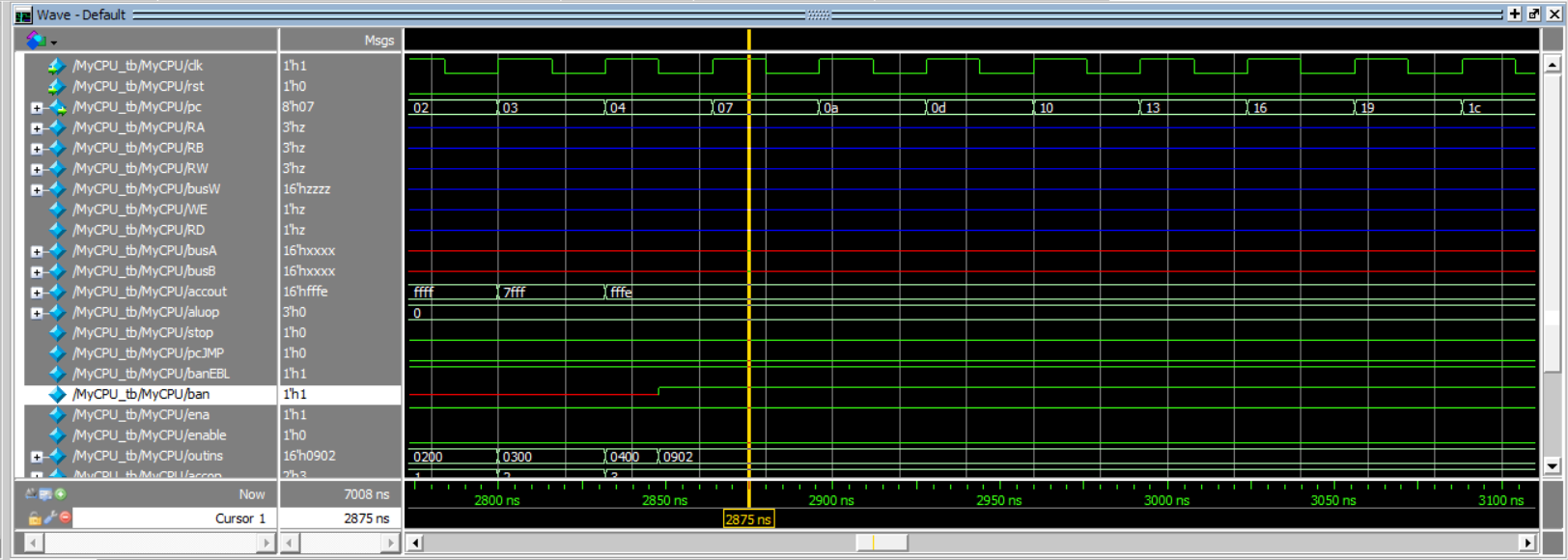


图 5.２２ 执行有条件转移（负则转）指令BAN后 1

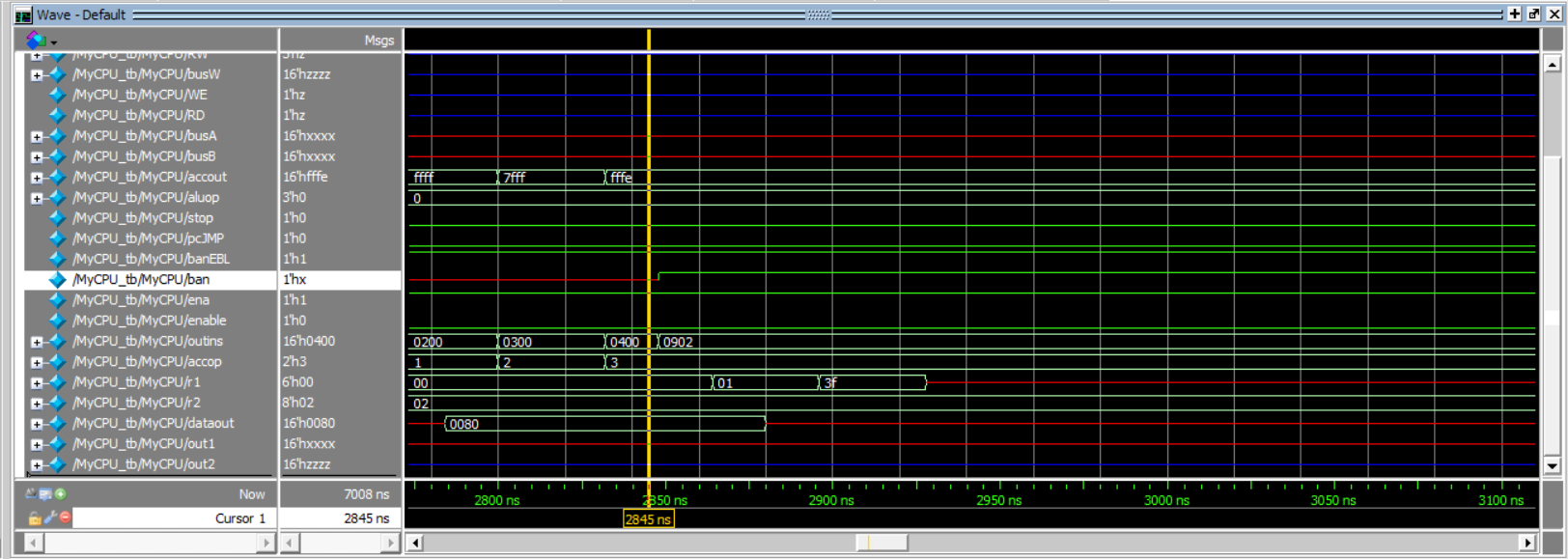


图 5.２３ 执行有条件转移（负则转）指令BAN前 2

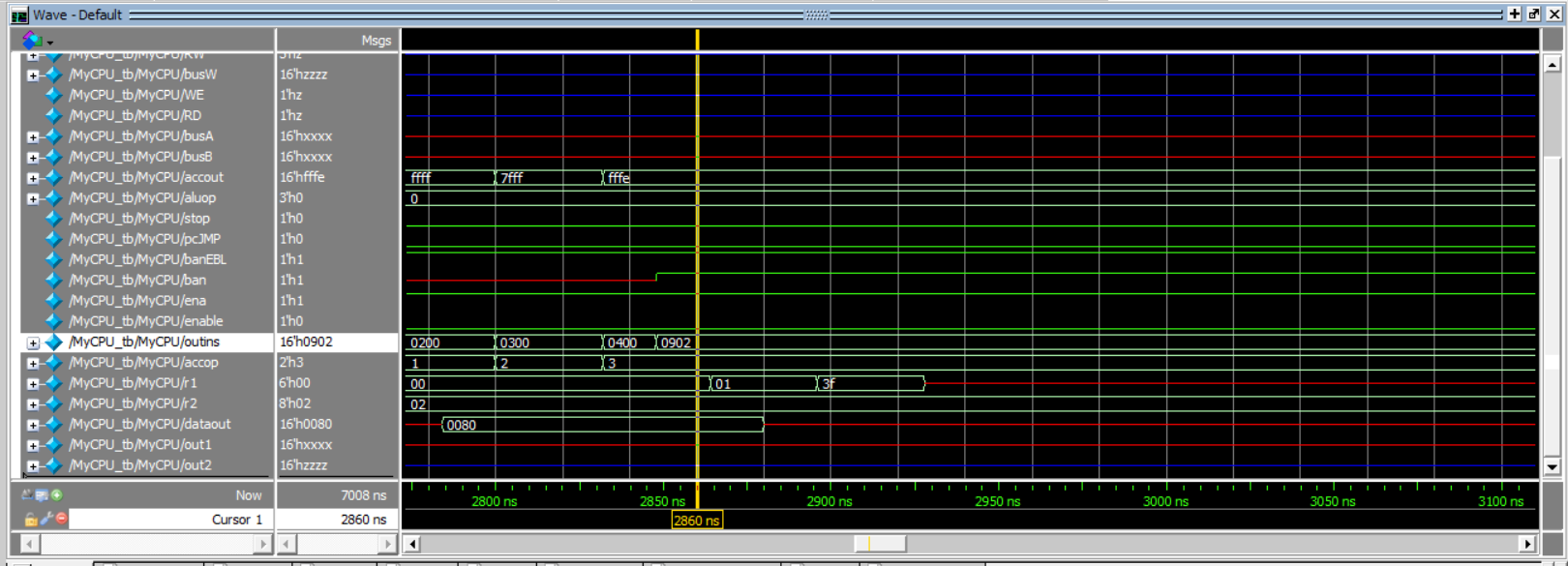


图 5.２４ 执行有条件转移（负则转）指令BAN后 2

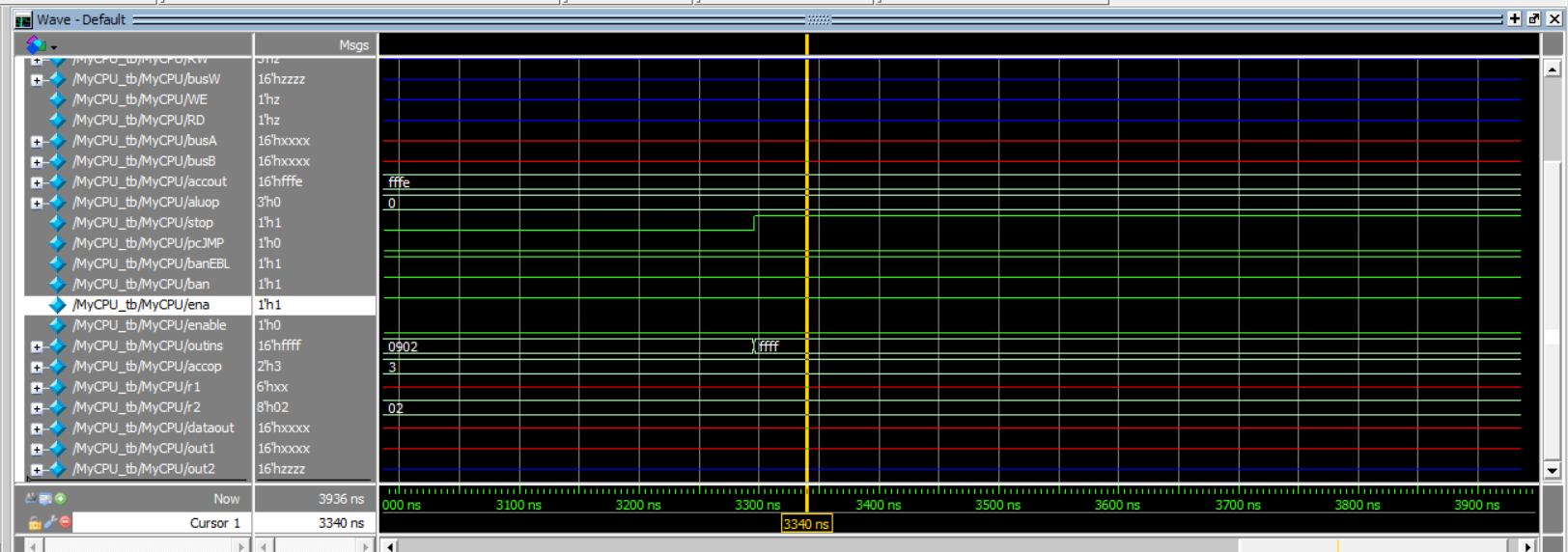


图 5.２５ 执行停机指令STP后 1

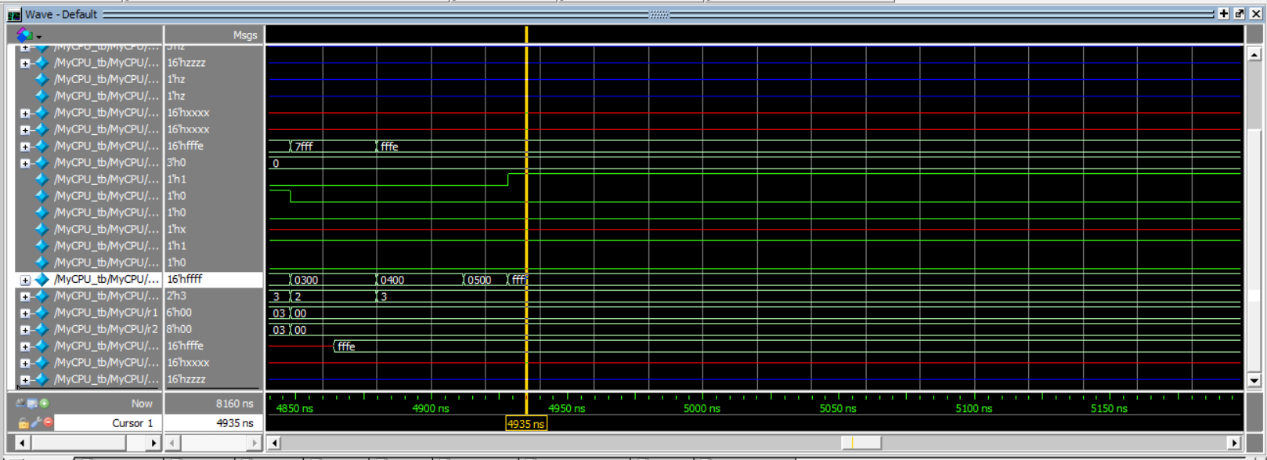


图 5.２６ 执行停机指令STP后 2

# 实验结果与分析

1. 开机指令START：执行开机指令START前，信号stop为不确定状态，CPU处于停机状态；执行START后，stop跳变为0，说明此时系统意见开始工作。如图5.1与图5.2所示。
2. 清除累加器指令CLA：执行清除累加器指令CLA前，程序计数寄存器ACC为不确定状态，ACC处于待机状态；执行CLA后，0000被送入ACC中，CPU可以开始执行程序。如图5.3与图5.4所示。
3. 累加器取反指令COM：执行累加器取反指令COM前，ACC的输出accout为0000H；执行后COM后，accout跳变为FFFFH，ACC的内容被成功取反输出。如图5.5与图5.6所示。
4. 算术右移一位指令SHR：执行算术右移一位指令SHR前，ACC的输出accout为FFFFH；执行SHR后，accout跳变为7FFFH，ACC的内容被成功右移一位。如图5.7与图5.8所示。
5. 循环左移一位指令CSL：执行循环左移一位指令CSL前，ACC的输出accout为7FFFH；执行CLS后，accout跳变为FFFEH，ACC的内容被成功循环左移一位。如图5.9与图5.10所示。
6. 加法指令ADD X：执行加法指令ADD X前，程序计数器寄存器ACC的值为9H，r1为5H，r2为9H；执行ADD X后，ACC的值为13H，即(r2)+(ACC)->ACC,ACC+1->ACC。如图5.11与图5.12所示。
7. 存数指令STA X：执行存数指令STA X前，存储器dataout为不确定状态；执行STA X后，因ACC=06H，传送給ALU的指令为0B，因此ACC中的数据FFFEH在下一个时钟下降沿传送給dataout。如图5.13与图5.14所示。
8. 取数指令LDA X：执行取数指令LDA X前，程序计数器寄存器ACC的值为06H；执行LDA X后，因accop=3H，所以依然执行(ACC)+1->ACC，因此ACC的值在下一个时钟下降沿变为07H。如图5.15与图5.16所示。
9. 无条件转移指令JMP imm：执行无条件转移指令JMP imm前，程序计数器寄存器ACC的值为07H，pcJMP=1H，允许无条件跳转；执行JMP imm后，outins=0803H，ins=03H，所以在下一个时钟上升沿，ACC跳变到03H。如图5.17至5.20所示。
10. 有条件转移（负则转）指令BAN X：执行有条件转移（负则转）指令BAN X前，程序计数器寄存器ACC的值为04H，r2=02H，ban=1，允许有条件跳转，；执行有条件转移（负则转）指令BAN X后，(ACC)+(r2)->ACC，(ACC)+1->ACC，所以在下一个时钟上升沿，ACC跳变为07H。如图5.21至图5.26所示。

所有指令均按预定程序完成运算。

# 心得体会

Verilog虽然跟C语言和Pascal语言很像，但还是有不少差别的，使用起来有些困难。加之对CPU的构造并不十分熟悉，导致在程序调试过程中出了很多bug。另外，必须承认的是，这个实验比较难，在实现过程中有参考网上的资源。

ModelSim真难用！刚开始我用的是ModelSim 10.1，非常卡顿。我猜测是因为32位软件与64位系统不兼容的原因，遂换了ModelSim SE 2019.2，还是非常卡顿，UI优化做得很垃圾，对鼠标点击和键盘的响应非常慢。做课设绝对不会用这个软件，太影响开发心情了。