“计算机设计与实践”处理器实验设计报告

姓名：许家乐

班级：1503103

学号：1150310329

哈尔滨工业大学计算机学院

2017 年 7 月

摘要

在本学期的《计算机设计实践》课程中，我们学习了 VHDL 硬件描述语言和 FPGA 编程的相关知识，并对《计算机组成原理》中的知识进行了回顾。通过本课程的学习，我对 VHDL 语言掌握得更加透彻，对 FPGA 编程思想有了更深入的了解，同时也对计算机系统的结构组成理解得更加清晰透彻，可谓收获颇丰。

关键字：VHDL CPU FPGA 层次化系统设计

目录

一．实验概述

1.实验目的

①掌握 Xiilinx ISE 集成开发环境的使用方法。
②掌握 VHDL 语言。
③掌握 FPGA 编程方法及硬件调试手段。
④深刻理解处理器结构和计算机系统的整体工作原理。

2.实验环境

①Xilinx ISE 14.7 集成开发环境
②Isim 仿真工具
③SD2100 数字逻辑设计实验平台

3.实验内容

根据计算机组成原理课程所学的知识和本课程所讲的设计思想，设计并实现一个给定指令系统的处理器，包括 VHDL 语言的实现和 FPGA 芯片的编程实现，为以后应用和设计处理器系统打下基础。

二．处理器设计方案

1.接口信号定义

信号名	位数	方向	来源/去向	意义
RST	1	I	处理器板	高电平复位信号
CLK	1	I	处理器板	系统时钟
ABus	16	O	主存储器	地址总线
DBus	16	I/O	主存储器	数据总线
nMREQ	1	O	主存储器	存储器片选信号
nRD	1	O	主存储器	存储器读信号
nWR	1	O	主存储器	存储器写信号
nBHE	1	O	主存储器	高字节访问允许
nBLE	1	O	主存储器	低字节访问允许
nPREQ	1	O	外设	端口片选信号

nPRD	1	O	外设	端口读有效
nPWR	1	O	外设	端口写有效
IOAD	2	O	外设	端口地址
IOR0	8	I	拨码开关	输入端口 0
IOR1	8	I	拨码开关	输入端口 1
IOR2	8	I	拨码开关	输入端口 2
IOR3	8	I	拨码开关	输入端口 3
IOW0	8	O	数码管	输出端口 0
IOW1	8	O	数码管	输出端口 1
IOW2	8	O	数码管	输出端口 2
IOW3	8	O	数码管	输出端口 3
IRtest	16	O	数码管	IR 输出观察
Ttest	4	O	LED 指示灯	节拍输出观察

2.指令格式设计

2.1 指令格式

寄存器——寄存器型指令：

OP	AD1		AD2

OP：操作码，5 位

AD1：寄存器地址 1，3 位

AD2：寄存器地址 2，3 位

其他类型指令：

OP	AD1	X

OP：操作码，5 位

AD1：寄存器地址，3 位

X：形式地址，8 位

各指令格式设计：

指令名称	指令格式	解释
JMP	XXXXX; XXX; XXXXXXXX	操作码；无效数据；形式地址
JZ	XXXXX; XXX; XXXXXXXX	操作码；寄存器地址；形式地址
SUB	XXXXX; XXX; XXXXX; XXX	操作码；寄存器地址；无效数据；寄存器地址
ADD	XXXXX; XXX; XXXXX; XXX	操作码；寄存器地址；无效数据；寄存器地址
MVI	XXXXX; XXX; XXXXXXXX	操作码；寄存器地址；立即数
MOV	XXXXX; XXX; XXXXX; XXX	操作码；寄存器地址；无效数据；寄存器地址
STA	XXXXX; XXX; XXXXXXXX	操作码；寄存器地址；形式地址
LDA	XXXXX; XXX; XXXXXXXX	操作码；寄存器地址；形式地址
OUT	XXXXX; XXX; XXXXXX; XX	操作码；寄存器地址；无效数据；端口地址
IN	XXXXX; XXX; XXXXXX; XX	操作码；寄存器地址；无效数据；端口地址

2.2 操作码分配

指令名称	指令助记符	操作码
无条件跳转	JMP	00000
条件跳转	JZ	00010
减法操作	SUB	00100
加法操作	ADD	00110
立即数传送	MVI	01000
寄存器传送	MOV	01010
存数操作	STA	01100
取数操作	LDA	01110
输出操作	OUT	10000
输入操作	IN	10010

3.微操作的定义

定义：

Ad1(IR)：IR 的 8-10 位，表示寄存器地址

Ad2(IR)：IR 的 0-2 位，表示寄存器地址

Ad(IR)：IR 的 0-7 位，表示形式地址

取指阶段：

PC → MAR，1 → R，M(MAR) → MDR，MDR → IR，PC + 1 → PC

执行阶段：

①非访存指令

ADD Ri, Rj：

Reg(Ad1(IR)) + Reg(Ad2(IR)) → Reg(Ad1(IR))

SUB Ri, Rj：

Reg(Ad1(IR)) - Reg(Ad2(IR)) → Reg(Ad1(IR))

MOV Ri, Rj：

Reg(Ad2(IR)) → Reg(Ad1(IR))

MVI Ri, X：

X → Reg(Ad1(IR))

②访存指令

STA Ri, X：

Reg(R7) // Ad(IR) → MAR, 1 → W,

Reg(Ad1(IR)) → MDR,

MDR → M(MAR)

LDA Ri, X：

Reg(R7) // Ad(IR) → MAR, 1 → R,

M(MAR) → MDR,

MDR → Reg(Ad1(IR))

③转移类指令

JZ Ri, X：

Zero(Reg(Ad1(IR))) * Reg(R7) // Ad(IR) + NZero(Reg(Ad1(IR))) * PC → PC

JMP X：

Reg(R7) // Ad(IR) → PC

④I/O 指令

IN Ri, Port：

Ad2(IR) → MAR, 1 → R,

M(MAR) → MDR,

MDR → Reg(Ad1(IR))

OUT Ri, Port：

Ad2(IR) → MAR, 1 → W,

Reg(Ad1(IR)) → MDR,

MDR → M(MAR)

4.节拍的划分

每个机器周期包含 4 个机器周期：

①取指周期

②运算周期

③访存周期

④回写周期

每个机器周期包含 1 个节拍。

节拍	周期	工作模块	说明
T0	取指周期	取指管理模块	完成取指令操作
T1	运算周期	运算管理模块	PC 加 1，进行指令译码，计算操作数和有效地址
T2	访存周期	存储管理模块	完成读写主存和外设的输入输出操作
T3	回写周期	回写管理模块	根据控制信号将数据回写到通用寄存器或 PC 中

5．处理器结构设计框图及功能描述

5.1 功能描述

①该处理器在给定的指令集下构建，支持十条指令。

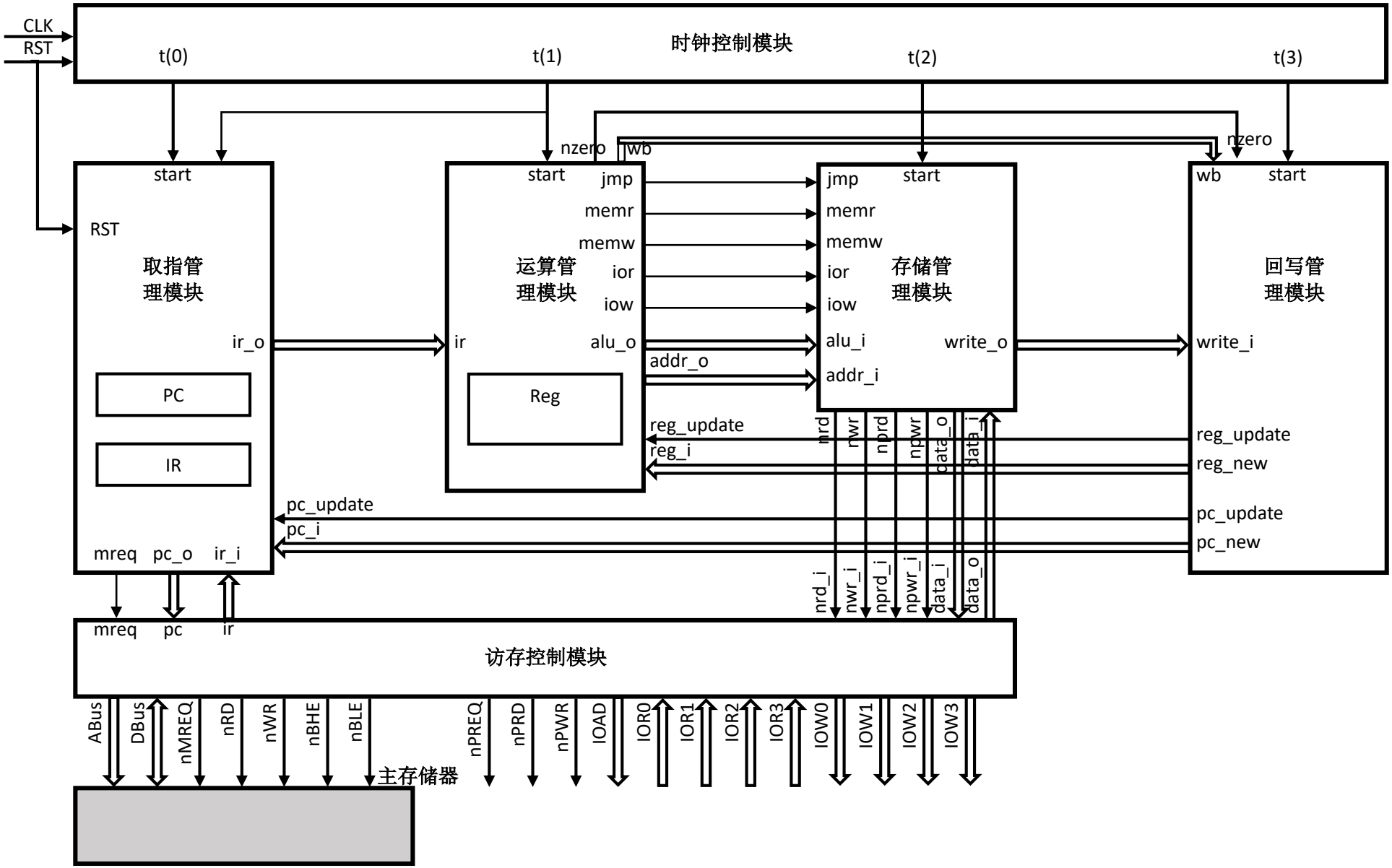
②假定主存可在一个时钟周期内完成一次存取操作，且可和 CPU 同步工作。

③系统使用一个主存单源，指令读取和数据访问使用同一组存储器。

④处理器指令字长 16 位，包含 8 个 8 位通用寄存器，1 个 16 位指令寄存器 IR，一个 16 位程序计数器 PC。

⑤取指令时，可以直接从主存提取 16 位的指令信息，而进行数据访问时，与主存进行 8 位的数据交换。处理器使用 16 位地址总线和数据总线，无论是取指还是数据访问都使用同一组数据总线，只是信息宽度不同。

5.2 设计框图


CLK
RST
时钟控制模块
t(0)
t(1)
t(2)
t(3)
nzero
wb
start
RST
取指管
理模块
ir_o
PC
IR
mreq
pc_o
ir_i
运算管
理模块
ir
Reg
jmp
memr
memw
ior
iow
alu_o
addr_o
reg_update
reg_i
pc_update
pc_i
存储管
理模块
alu_i
addr_i
write_o
nrd
nwr
nprd
npwr
data_o
data_i
回写管
理模块
write_i
reg_update
reg_new
pc_update
pc_new
nrd_i
nwr_i
nprd_i
npwr_i
data_i
data_o
mreq
pc
ir
访存控制模块
ABus
DBus
nMREQ
nRD
nWR
nBHE
nBLE
主存储器
nPREQ
nPRD
nPWR
IOAD
IOR0
IOR1
IOR2
IOR3
IOW0
IOW1
IOW2
IOW3

三．功能模块设计方案

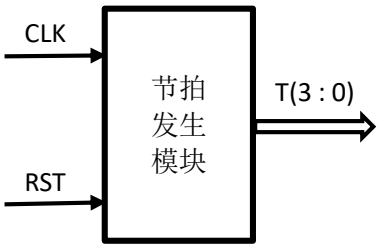
1.时钟控制模块

1.1 功能描述

通过系统时钟输入信号 CLK 产生节拍（4 拍循环），控制各模块的工作，4 个输出信号分别代表 4 个节拍。

RST 是系统复位信号，当 RST 为高电平时，所有输出信号均为‘0’；当 RST 为低电平时，每一个 CLK 信号的上升沿到来时，T 信号按照“0001”,“0010”,“0100”,“1000”的顺序循环输出，以产生节拍。

1.2 设计框图



1.3 接口信号

信号名	位数	方向	来源/去向	意义
RST	1	I	处理器板	高电平复位信号
CLK	1	I	处理器板	系统时钟
T	4	O	其他各管理模块	节拍输出

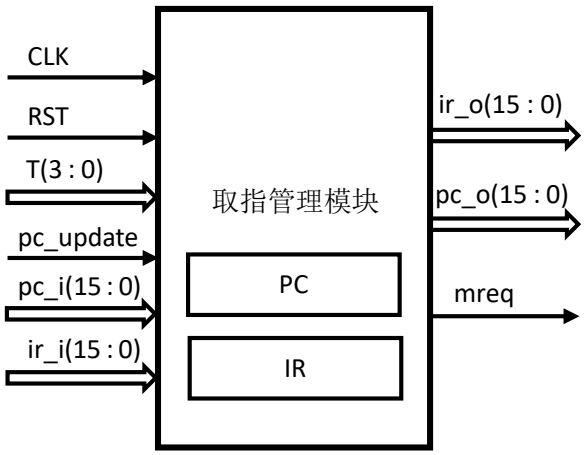
2.取指管理模块

2.1 功能描述

取指管理模块在取指周期（节拍 T(0)）工作，完成从主存中取指令的操作，并将取出的指令送入运算模块进行指令译码。此模块包含程序计数器 PC 和指令寄存器 IR，其中 PC 在运算周期的上升沿完成自动加 1。

当节拍 T(0)为高电平时,取指管理模块进入工作周期。访存请求信号 mreq 输出高电平，pc_o 输出信号将程序计数器 PC 中的内容送入访存控制模块，由其完成访存操作并通过 ir_i 输入信号将取出的指令存入指令寄存器 IR 中。当系统时钟 CLK 的一个下降沿到来，并且 T(0)仍为高电平时，ir_o 输出信号将指令寄存器 ir 中的内容输出并保持，以便其他模块进行后续操作。当节拍 T(1)的上升沿到来时，PC 自动加 1。在回写周期，若 pc_update 信号有效，则将 pc_i 信号的输入写入 PC 中，以达到程序跳转的目的。

2.2 设计框图



2.3 接口信号

信号名	位数	方向	来源/去向	意义
CLK	1	I	处理器板	系统时钟
RST	1	I	处理器板	高电平复位信号
T	4	I	时钟控制模块	节拍信号
pc_update	1	I	回写管理模块	PC 更新控制信号
pc_i	16	I	回写管理模块	PC 更新内容
ir_i	16	I	访存控制模块	取出的指令
ir_o	16	O	运算管理模块	当前指令
pc_o	16	O	访存控制模块	待取指令的地址
mreq	1	O	访存控制模块	取指访存请求信号

3.运算管理模块

3.1 功能描述

运算管理模块在运算周期（节拍 T(1)）工作，完成指令译码，操作数的准备和访存有效地址的计算。同时对条件跳转指令完成条件判断，并将结果输出作为控制信号。此模块包含 8 位通用寄存器组 Reg1-Reg7。

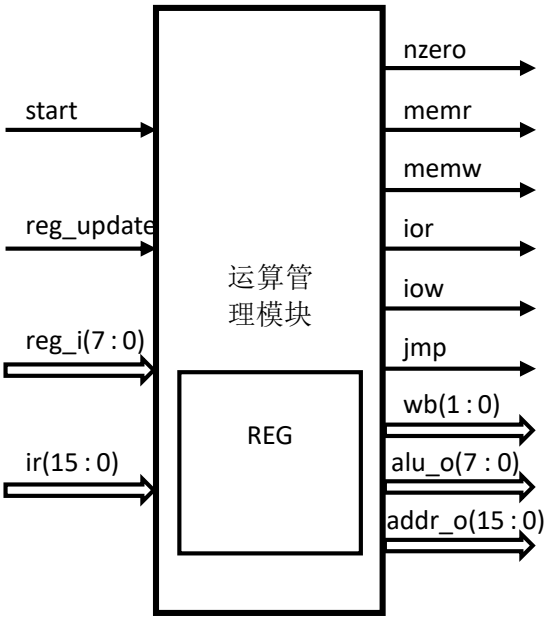
当 start 信号为高电平时，运算管理模块进入工作周期。ir 输入的高 5 位为操作码，进行指令译码并产生 memr，memw，ior，iow，jmp，wb 控制信号，同时根据操作码计算出该指令所需的操作数和有效地址，并将结果分别通过 alu_o 和 addr_o 进行输出。当指令第 10 到 8 位表示的寄存器中的内容为“00000000”时，nzero 信号输出‘0’；否则，nzero 输出‘1’。在回写周期，若 reg_update 信号有效，则将 reg_i 信号的输入写入指令第 10 到 8 位表示的寄存器中，以达到更新寄存器的目的。

各指令与控制信号的对应关系如下表：

指令名	memr	memw	ior	iow	jmp	wb
JMP	0	0	0	0	1	01
JZ	0	0	0	0	1	10
SUB	0	0	0	0	0	11
ADD	0	0	0	0	0	11
MVI	0	0	0	0	0	11
MOV	0	0	0	0	0	11

STA	0	1	0	0	0	00
LDA	1	0	0	0	0	11
OUT	0	0	0	1	0	00
IN	0	0	1	0	0	11

3.2 结构框图



3.3 接口信号

信号名	位数	方向	来源/去向	意义
start	1	I	时钟控制模块	模块使能信号
reg_update	1	I	回写管理模块	寄存器更新控制信号
reg_i	8	I	回写管理模块	寄存器更新内容
ir	16	I	取指管理模块	当前指令
nzero	1	O	回写管理模块	寄存器为 0 判断结果
memr	1	O	存储管理模块	主存读控制信号
memw	1	O	存储管理模块	主存写控制信号
ior	1	O	存储管理模块	外设读控制信号
iow	1	O	存储管理模块	外设写控制信号
jmp	1	O	存储管理模块	跳转控制信号
wb	2	O	回写管理模块	回写方式控制信号
alu_o	8	O	存储管理模块	操作数计算结果
addr_o	16	O	存储管理模块	有效地址计算结果

4.存储管理模块

4.1 功能描述

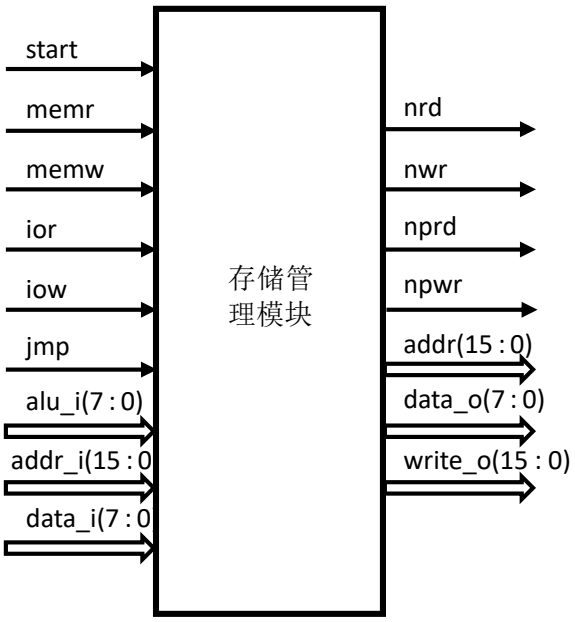
存储管理模块在访存周期（节拍 T(2)）工作，根据控制信号的不同，向访存控制模块发出主存读写请求和外设读写请求，并传递访问主存和外设所需的地址和数据。当需要回写时，根据控制信号的不同产生回写内容并将其传送给回写控制模块。

当 start 信号为高电平时，存储管理模块进入工作周期。若 memr 控制信号为高电平，其他读写控制信号为低电平，则输出信号 nrd 为低电平有效，表示向访存控制模块发出主存读请求；若 memw 控制信号为高电平，其他读写控制信号为低电平，则输出信号 nwr 为低电平有效，表示向访存控制模块发出主存写请求；若 ior 控制信号为高电平，其他读写控制信号为低电平，则输出信号 nprd 为低电平有效，表示向访存控制模块发出外设读请求；若 iow 控制信号为高电平，其他读写控制信号为低电平，则输出信号 npwr 为低电平有效，表示向访存控制信号发出外设写请求。

当读或写控制信号之一（即 memr,memw,ior,iow）为高电平时，addr 输出信号将 addr_i 地址输入信号的内容传送给访存控制模块作为访存地址；当写控制信号（即 memw,iow）之一为高电平时，data_o 输出信号将 alu_i 操作数输入信号的内容传送给访存控制模块作为待写入数据。

当读控制信号（即 memr,ior）之一为高电平时，write_o 低 8 位将访问主存或外设读出的数据 data_i 输出，传送给回写管理模块以便在回写周期更新通用寄存器；当读写控制信号均为低电平，且 jmp 跳转信号也为低电平时，表示指令进行算术运算或数据移动操作（SUB,ADD,MVI,MOV），write_o 低 8 位将来自运算管理模块的操作数 alu_i 输出，传送给回写管理模块以便在回写周期更新寄存器；当读写控制信号均为低电平，且 jmp 跳转信号为高电平时，表示指令进行跳转操作（JMP,JZ），write_o 将来自运算管理模块的有效地址 addr_i 输出，传送给回写管理模块以便在回写周期更新 PC。

4.2 设计框图



4.3 接口信号

信号名	位数	方向	来源/去向	意义
start	1	I	时钟控制模块	模块使能信号
memr	1	I	运算管理模块	主存读控制信号
memw	1	I	运算管理模块	主存写控制信号
ior	1	I	运算管理模块	外设读控制信号
iow	1	I	运算管理模块	外设写控制信号
jmp	1	I	运算管理模块	跳转控制信号
alu_i	8	I	运算管理模块	操作数计算结果
addr_i	16	I	运算管理模块	有效地址计算结果

data_i	8	I	访存控制模块	读取结果
nrd	1	O	访存控制模块	主存读请求
nwr	1	O	访存控制模块	主存写请求
nprd	1	O	访存控制模块	外设读请求
npwr	1	O	访存控制模块	外设写请求
addr	16	O	访存控制模块	读写地址
data_o	8	O	访存控制模块	待写数据
write_o	16	O	回写管理模块	待回写内容

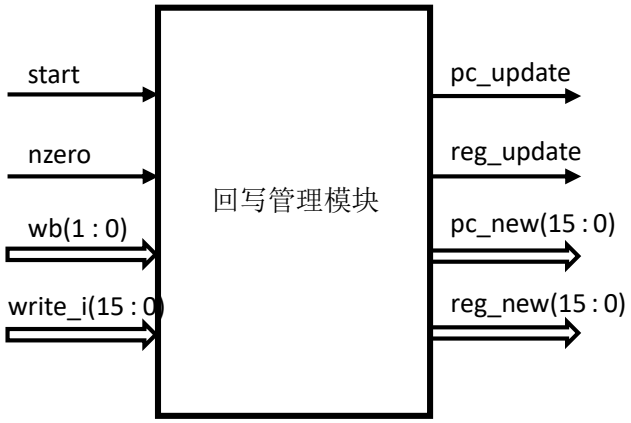
5.回写管理模块

5.1 功能描述

回写管理模块在回写周期（节拍 T(3)）工作，根据控制信号的不同，发出 PC 更新信号和通用寄存器更新信号，将来自存储管理模块的回写内容回写到 PC 或通用寄存器中。

当 start 信号为高电平时，回写管理模块进入工作周期。pc_new 输出信号表示需要回写到 PC 中的内容，reg_new 输出信号表示需要会写到通用寄存器中的内容，它们通过组合逻辑直接与 write_i 回写内容输入信号相连。若 wb 输入为“00”，则表示无需回写，pc_update 与 reg_new 信号均为高阻态；若 wb 输入为“01”，则表示需要不经过判断直接回写到 PC（对应 JMP 指令），此时 pc_update 为高电平有效，pc_new 输出 write_i；若 wb 输入为“10”，则表示需要经过判断后选择是否回写到 PC（对应 JZ 指令），此时若 nzero 信号为低电平，则表示寄存器内容为 0，进行回写，反之则不进行回写；若 wb 输入为“00”，则表示需要回写到通用寄存器，此时 reg_update 为高电平有效，reg_new 输出 write_i 的低 8 位。当回写周期结束时，pc_update 和 reg_update 信号立刻回到低电平状态。

5.2 设计框图



5.3 接口信号

信号名	位数	方向	来源/去向	意义
start	1	I	时钟控制模块	模块使能信号
nzero	1	I	运算管理模块	寄存器为 0 判断结果
wb	2	I	运算管理模块	回写方式控制信号
write_i	16	I	存储管理模块	待回写内容
pc_update	1	O	取指管理模块	PC 更新控制信号
reg_update	1	O	运算管理模块	寄存器更新信号
pc_new	16	O	取指管理模块	PC 更新内容

reg_new	8	O	运算管理模块	寄存器更新内容

6.访存控制模块

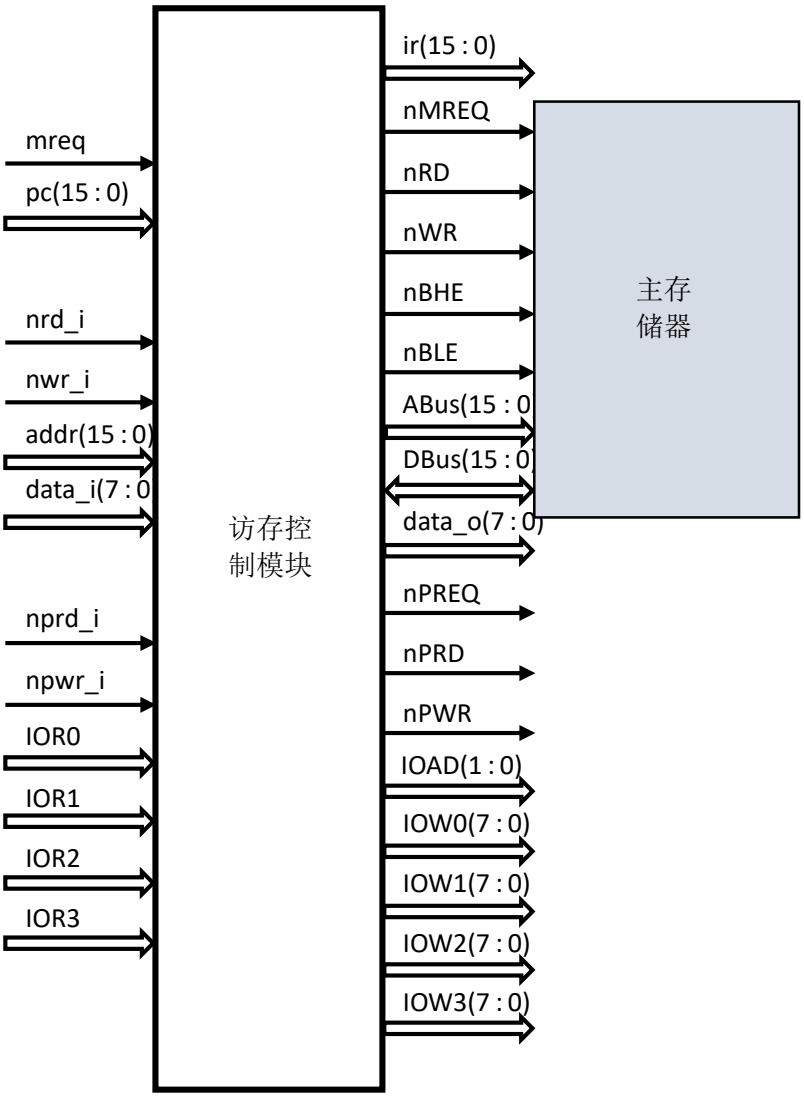
6.1 功能描述

访存控制信号在取指周期和访存周期工作。在取指周期接受来自取指管理模块的 PC 地址，访存取出指令后将其送入取值管理模块；在访存周期有主存或外设的读写请求时，对主存或外设进行相应的读写操作，并与存储管理模块进行数据交换。

当 mreq 信号为高电平时，表示此时有取指访存请求，访存控制模块将来自取指管理模块的 pc 输入信号内容输出到地址总线 ABus，同时 nMREQ、nRD、nBHE、nBLE 控制信号为低电平有效，当指令从主存读出后，将其从数据总线 DBus 输出到 ir 送入取指管理模块。

当 nrd_i 信号为低电平时，表示此时有主存读请求，访存控制模块将来自存储管理模块的地址 addr 输出到地址总线 ABus，同时 nMREQ、nRD、nBLE 控制信号为低电平有效，当数据从主存取出后，将其从数据总线 DBus 输出到 data_o 送入存储管理模块；当 nwr_i 信号为低电平时，表示此时有主存写请求，访存控制模块将来自存储管理模块的地址 addr 和数据 data_i 分别输出到地址总线 ABus 和数据总线 DBus，同时 nMREQ、nWR、nBLE 控制信号为低电平有效，将数据写入到主存；当 nprd_i 信号为低电平时，表示此时有外设读请求，访存控制模块根据地址 addr 的低 2 位从相应的输入端口中读取数据并输出到 data_o，将其送入存储管理模块；当 npwr_i 信号为低电平时，表示此时有外设写请求，访存控制模块根据地址 addr 的低 2 位将 data_i 中的数据输出到相应的输出端口。

6.2 结构框图



6.3 接口信号

信号名	位数	方向	来源/去向	意义
mreq	1	I	取指管理模块	取指访存请求
pc	16	I	取指管理模块	待取指令的地址
nrd_i	1	I	存储管理模块	主存读请求
nwr_i	1	I	存储管理模块	主存写请求
addr	16	I	存储管理模块	读写地址
data_i	8	I	存储管理模块	待写数据
nprd_i	1	I	存储管理模块	外设读请求
npwr_i	1	I	存储管理模块	外设写请求
IOR0	8	I	拨码开关	输入端口 0
IOR1	8	I	拨码开关	输入端口 1
IOR2	8	I	拨码开关	输入端口 2
IOR3	8	I	拨码开关	输入端口 3
ir	16	O	取指管理模块	当前指令
nMREQ	1	O	主存储器	存储器片选信号
nRD	1	O	主存储器	存储器读信号
nWR	1	O	主存储器	存储器写信号
nBHE	1	O	主存储器	高字节访问允许
nBLE	1	O	主存储器	低字节访问允许
ABus	16	O	主存储器	地址总线
DBus	16	I/O	主存储器	数据总线
data_o	8	O	存储管理模块	读出数据
nPREQ	1	O	外设	端口片选信号
nPRD	1	O	外设	端口读有效
nPWR	1	O	外设	端口写有效
IOAD	2	O	外设	端口地址
IOW0	8	O	数码管	输出端口 0
IOW1	8	O	数码管	输出端口 1
IOW2	8	O	数码管	输出端口 2
IOW3	8	O	数码管	输出端口 3

四．各模块及系统仿真

1.时钟控制模块

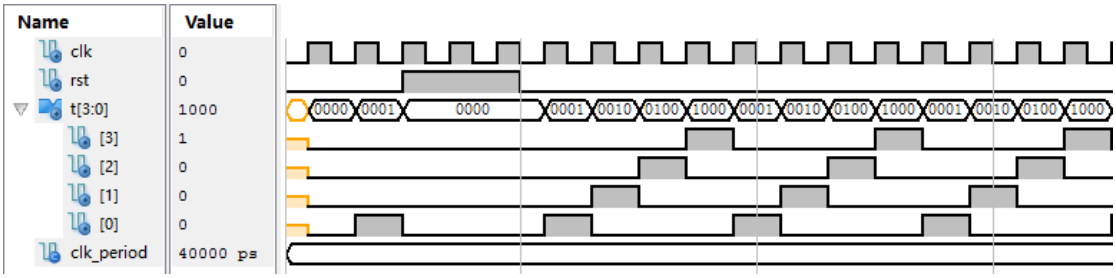
1.1 测试方案

①首先使 rst 信号无效，观察 t 信号的输出。
②然后使 rst 信号有效并保持一段时间，观察 t 输出是否恒为“0000”。
③再试 rst 信号无效，观察 t 是否从第一个时钟上升沿到来时开始产生节拍输出。

1.2 测试过程

```
stim_proc: process
   begin
      wait for 100 ns;
      rst <= '1';
      wait for 100 ns;
      rst <= '0';
      wait;
   end process;
```

1.3 仿真波形



由波形可知，当 rst 为高电平时，t 输出信号恒置为“0000”；当 rst 为低电平时，t 输出信号按照“0001”，“0010”，“0100”，“1000”的顺序循环输出，产生节拍。

2.取指管理模块

2.1 测试方案

①首先使 rst 输入信号有效,经过 100ns 后再使其无效，完成程序计数器 PC 内容的初始化。
②以 400ns 为一个指令周期，在每个周期内对 t 信号输入进行赋值，以模拟节拍信号。
③每当 t(0)信号有效时，通过 ir_i 信号进行指令输入，观察能否被 ir_o 信号输出并保持，同时观察 PC 值是否完成了自动加 1。
④在某个指令周期的最后一个节拍使 pc_update 信号有效并保持 100ns（即一个节拍的宽度），同时对 pc_i 输入信号赋值，观察在下一个指令周期 t(0)有效时 pc_o 输出是否变为所赋的值，以测试跳转功能。

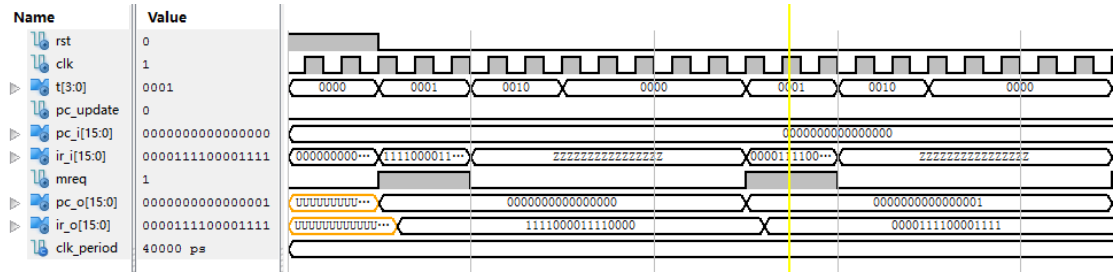
2.2 测试过程

```
stim_proc: process
   begin
      rst <= '1';
        wait for 100 ns;

        rst <= '0';
        t(0) <= '1';
        ir_i <= "1111000011110000";
        wait for 100 ns;

        t(0) <= '0';
        t(1) <= '1';
        ir_i <= (others=>'Z');
        wait for 100 ns;

        t(1) <= '0';
        wait for 200 ns;

        t(0) <= '1';
        ir_i <= "0000111100001111";
        wait for 100 ns;

        t(0) <= '0';
        t(1) <= '1';
        ir_i <= (others=>'Z');
        wait for 100 ns;

        t(1) <= '0';
        wait for 200 ns;

        t(0) <= '1';
        ir_i <= "1111000000001111";
        wait for 100 ns;

        t(0) <= '0';
        t(1) <= '1';
        ir_i <= (others=>'Z');
        wait for 100 ns;

        t(1) <= '0';
        wait for 100 ns;

        pc_update <= '1';
        pc_i <= "0000000011111111";
        wait for 100 ns;

        t(0) <= '1';
        pc_update <= '0';
        pc_i <= (others=>'Z');
        ir_i <= "0000111111110000";
        wait for 100 ns;

        t(0) <= '0';
```

```
        t(1) <= '1';
        ir_i <= (others=>'Z');
        wait for 100 ns;

        t(1) <= '0';
        wait for 200 ns;

        t(0) <= '1';
        ir_i <= "1111111100000000";
      wait for 100 ns;

        t(0) <= '0';
        t(1) <= '1';
        ir_i <= (others=>'Z');

      wait;
   end process;
```

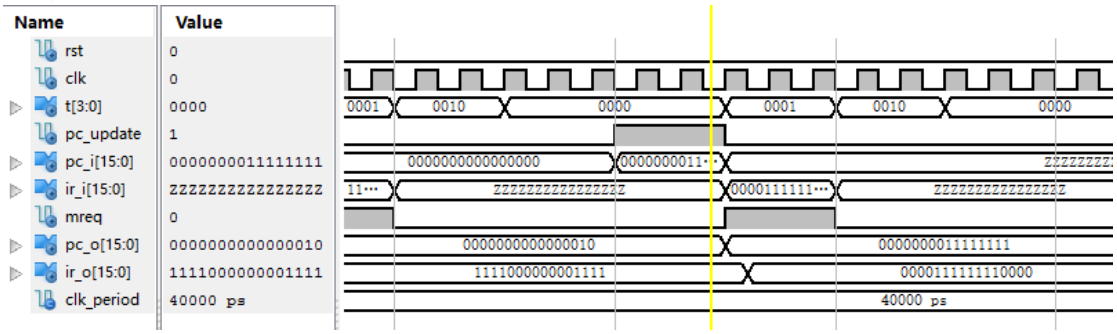
2.3 仿真波形

取指功能测试：



由仿真波形可知输入的指令在取指周期的 CLK 时钟下降沿到来时被正确输出，并在整个指令周期内保持不变；同时 PC 也完成了自动加 1；访存请求信号 mreq 在取指周期内有效。因此取指管理模块的取指令功能正常。

跳转功能测试：



由仿真波形可知当 pc_update 信号有效时，回写入 PC 的内容 pc_i 在下一个指令周期的取指节拍通过 pc_o 信号成功输出，因此取指管理模块的跳转功能正常。

3.运算管理模块

3.1 测试方案

①输入指令“0100011100110011”（MVI R7，00110011），观察各输出信号。
②输入指令“0100011011001100”（MVI R6，11001100），观察各输出信号。

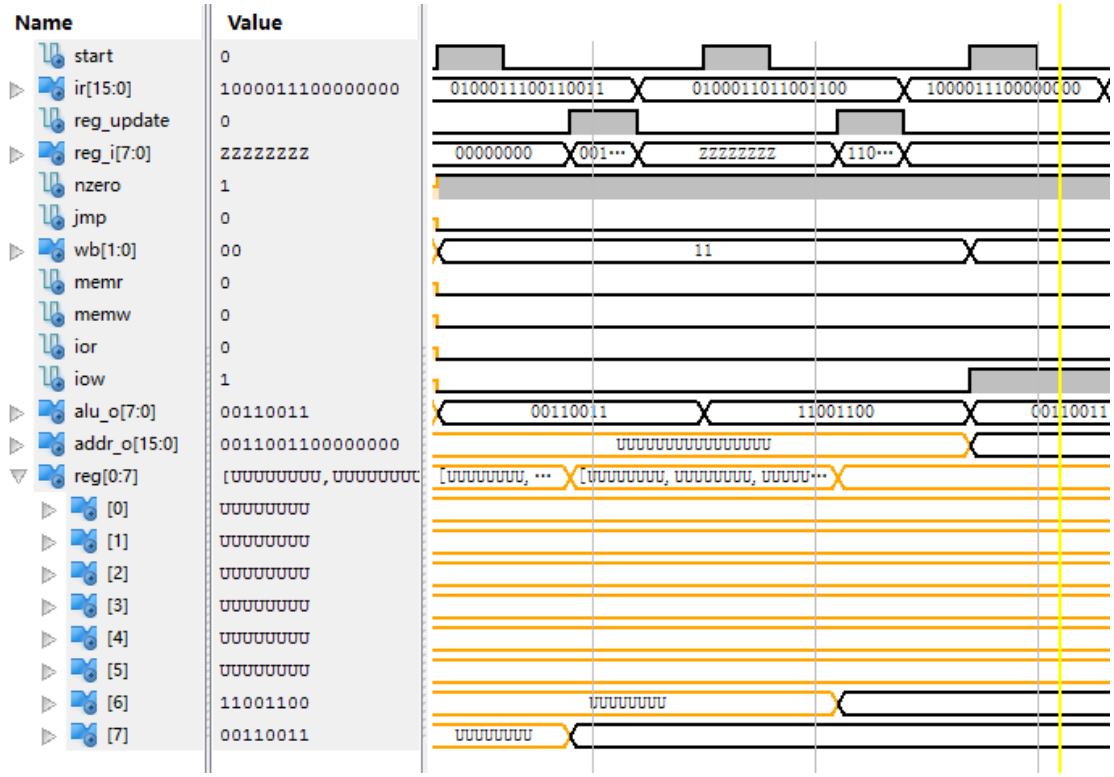
③输入指令“1000011100000000”（OUT R7，00），观察各输出信号。
④输入指令“0110011000000000”（STA R6, 0011001100000000），观察各输出信号。
⑤输入指令“0111001100000000”（LDA R6, 0011001100000000），观察各输出信号。
⑥输入指令“0011011000000111”（ADD R6, R7），观察各输出信号。
⑦输入指令“0000000000001111”（JMP 0011001100001111），观察各输出信号。

3.2 测试过程

```
stim_proc: process
   begin
        ir <= "0100011100110011"; --MVI R7, 00110011
      wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 60 ns;
        reg_update <= '1';
        reg_i <= "00110011";
        wait for 60 ns;

        reg_update <= '0';
        reg_i <= (others=>'Z');
        ir <= "0100011011001100"; --MVI R6, 11001100
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 60 ns;
        reg_update <= '1';
        reg_i <= "11001100";
        wait for 60 ns;

        reg_update <= '0';
        reg_i <= (others=>'Z');
        ir <= "1000011100000000"; --OUT R7, 00000000
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 60 ns;

        ir <= "0110011000000000"; --STA R6, 00000000
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 120 ns;

        ir <= "0111001100000000"; --LDA R6, 00000000
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 120 ns;

        ir <= "0011011000000111"; --ADD R6, R7
```

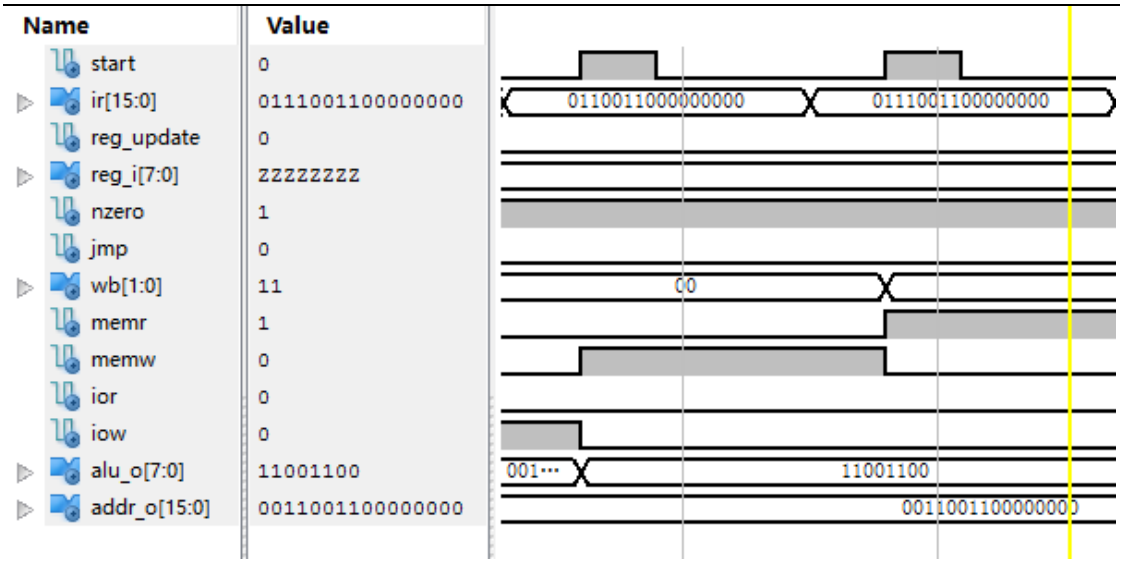
```
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 120 ns;

        ir <= "0000000000001111"; --JMP 0011001100001111;
        wait for 60 ns;
        start <= '1';
        wait for 60 ns;
        start <= '0';
        wait for 120 ns;

        wait;
    end process;
```

3.3 仿真波形



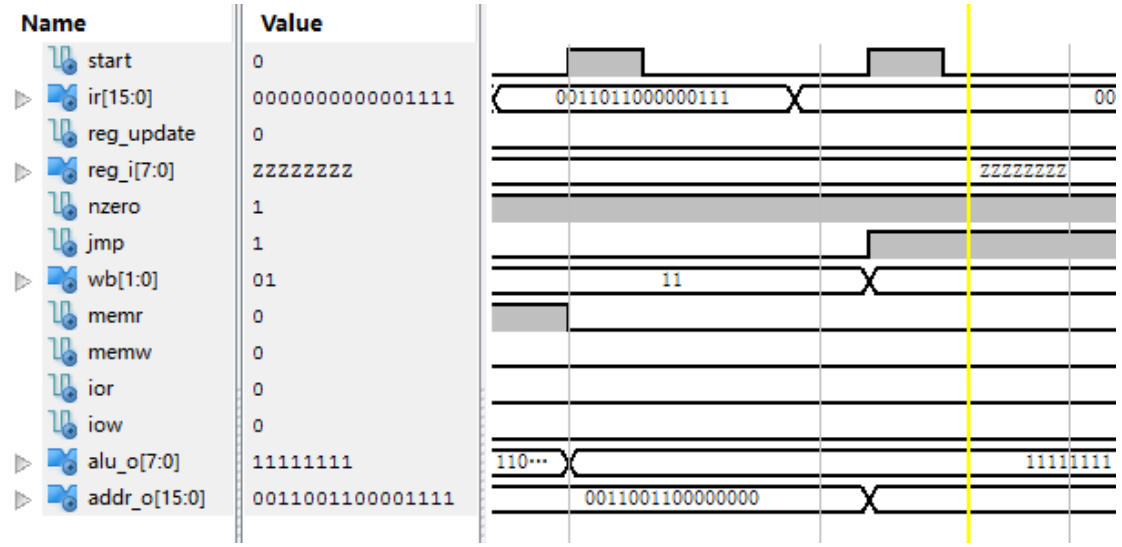
MVI R7，00110011　　MVI R6，11001100　　OUT R7，00：

由仿真波形可知在第一、二个指令周期内，内部信号 reg 的内容被成功更新，无读写控制信号发出，回写方式为“11”（即寄存器回写），alu_o 输出为指令的低八位（即立即数）；在第三个指令周期内，外设写控制信号 iow 有效，回写方式为“00”（无需回写），alu_o 输出为 reg7 寄存器中的内容 00110011，addr_o 输出为 reg7//X 即 0011001100000000。



STA R6, 0011001100000000　　LDA R6, 0011001100000000：

由仿真波形可知在 STA 指令周期内，存储器写信号 memw 有效，回写方式为“00”（无需回写），alu_o 输出为 reg6 中的内容 11001100，addr_o 输出为 reg7//X 即 0011001100000000；在 LDA 指令周期内，存储器读信号 memr 有效，回写方式为“11”（回写到寄存器），addr_o 输出为 reg7//X 即 0011001100000000。



ADD R6, R7　　JMP 0011001100001111：

由仿真波形可知在 ADD 指令周期内，无读写控制信号发出，回写方式为“11”（寄存器回写），alu_o 输出为 reg6 与 reg7 内容相加即 11111111；在 JMP 指令周期内，无读写控制信号发出，跳转控制信号 jmp 有效，回写方式为“01”（直接回写到 PC），addr_o 输出为 reg7//X 即 0011001100001111。

4.存储管理模块

4.1 测试方案

①令存储器读控制信号有效，模拟 LDA 指令控制信号，观察各输出信号。
②令存储器写控制信号有效，模拟 STA 指令控制信号，观察各输出信号。
③令外设读控制信号有效，模拟 IN 指令控制信号，观察各输出信号。

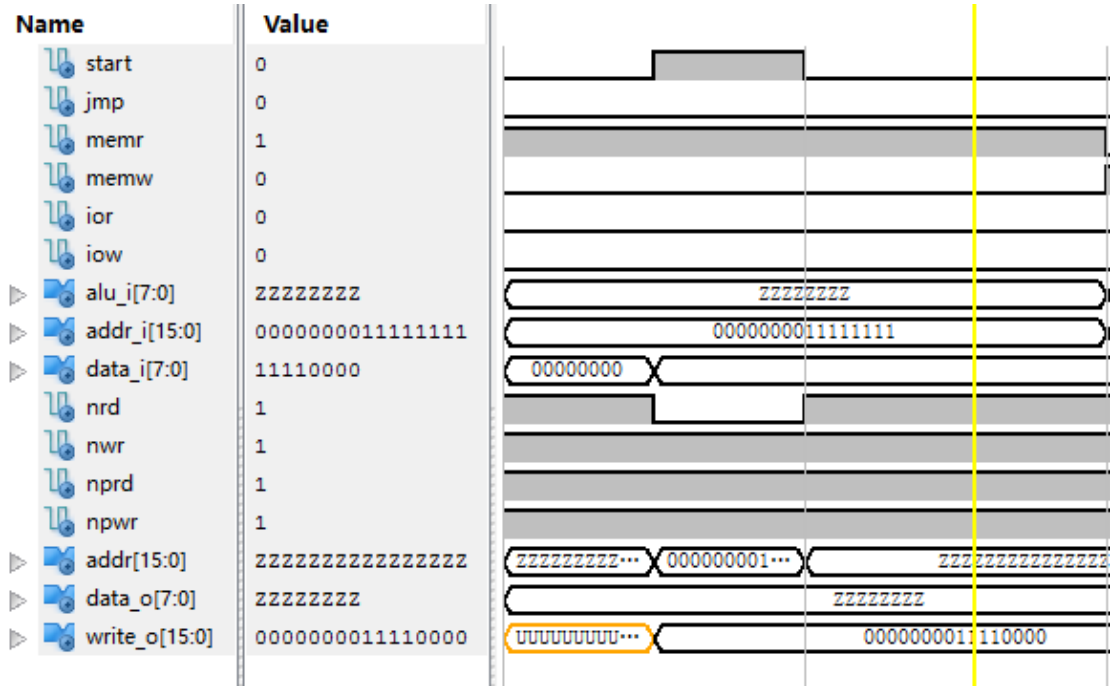
④令外设写控制信号有效，模拟 OUT 指令控制信号，观察各输出信号。

4.2 测试过程

```
stim_proc: process
   begin
      start <= '0';
        jmp <= '0';
        memr <= '1';
        memw <= '0';
        ior <= '0';
        iow <= '0';
        alu_i <= (others=>'Z');
        addr_i <= "0000000011111111";
        wait for 100 ns;

      start <= '1';       --LDA
        data_i <= "11110000";
      wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        jmp <= '0';
        memr <= '0';
        memw <= '1';
        ior <= '0';
        iow <= '0';
        alu_i <= "00110011";
        addr_i <= "0000111100001111";
        wait for 100 ns;

        start <= '1';     --STA
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        jmp <= '0';
        memr <= '0';
        memw <= '0';
        ior <= '1';
        iow <= '0';
        alu_i <= (others=>'Z');
        addr_i <= "1111000011110000";
        wait for 100 ns;

        start <= '1';     --IN
        data_i <= "10000001";
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        jmp <= '0';
        memr <= '0';
```

```
        memw <= '0';
        ior <= '0';
        iow <= '1';
        alu_i <= "10101010";
        addr_i <= "0011001100110011";
        wait for 100 ns;

        start <= '1';      --OUT
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        jmp <= '1';
        memr <= '0';
        memw <= '0';
        ior <= '0';
        iow <= '0';
        alu_i <= "10011001";
        addr_i <= "0100010001000100";
        wait for 100 ns;

        start <= '1';      --JMP
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

      wait;
   end process;
```

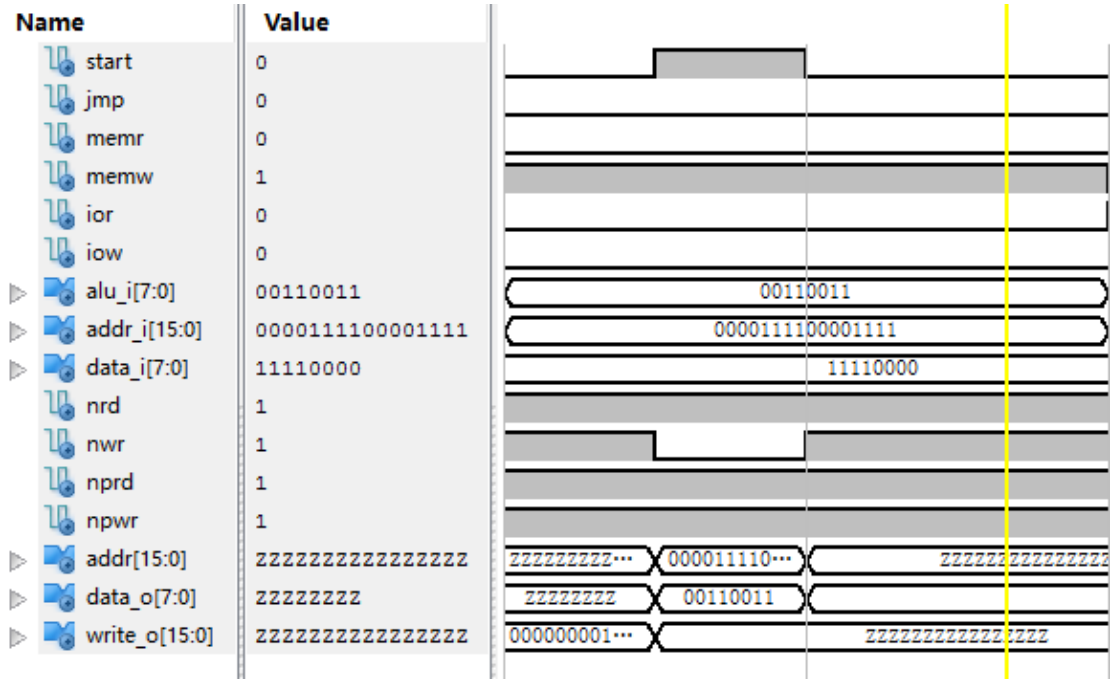
4.3 仿真波形



LDA：

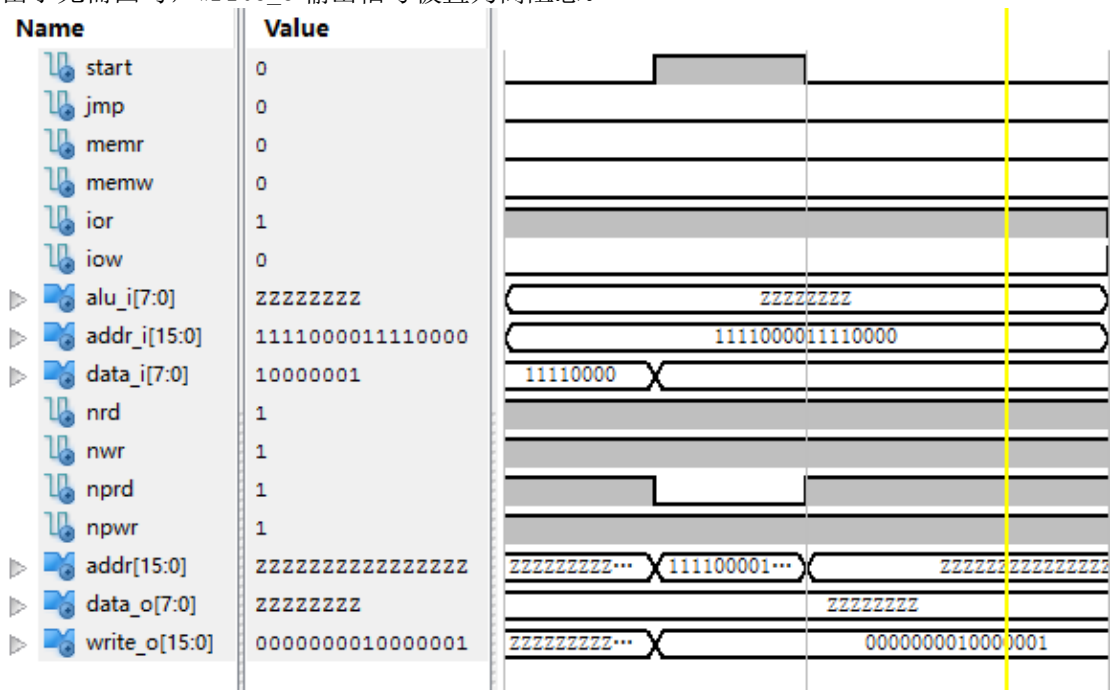
由仿真波形可知在 LDA 指令周期内，在访存周期时存储器读请求 nrd 低电平有效，同时

addr 输出来自 addr_i 的地址 0000000011111111，write_o 的低八位输出来自 data_i 的数据输入 11110000，高八位置为“00000000”。



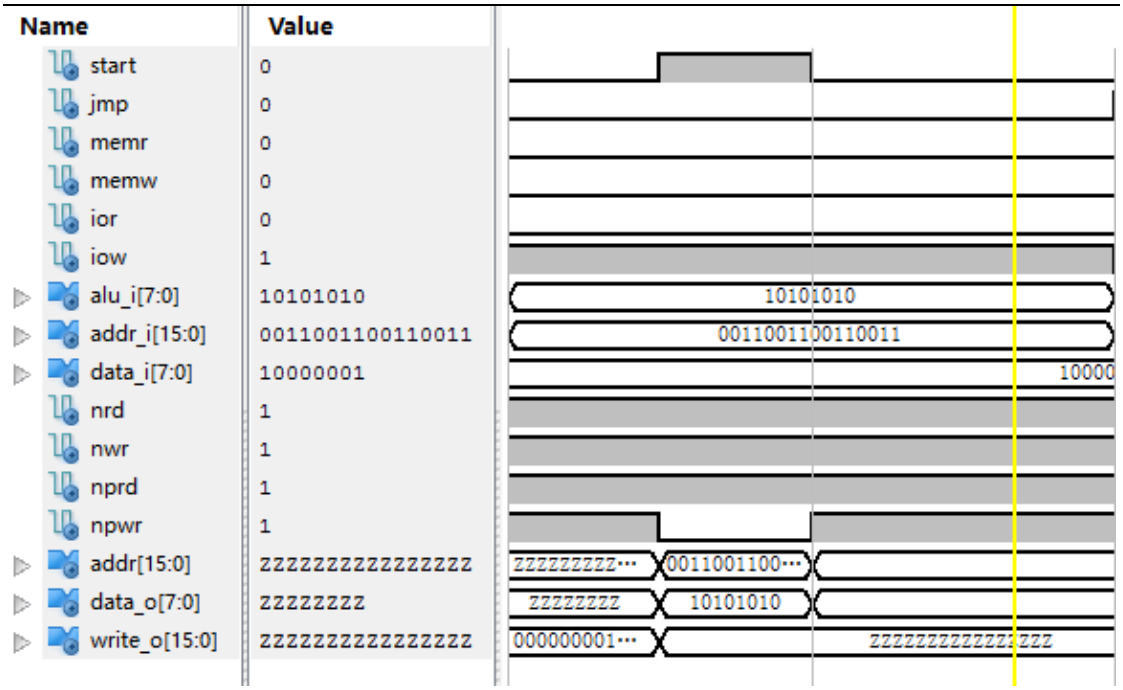
STA：

由仿真波形可知在 STA 指令周期内，在访存周期时存储器写请求 nwr 低电平有效，同时 addr 输出来自 addr_i 的地址 0000111100001111，data_o 输出来自 alu_i 的数据 00110011，由于无需回写，write_o 输出信号被置为高阻态。



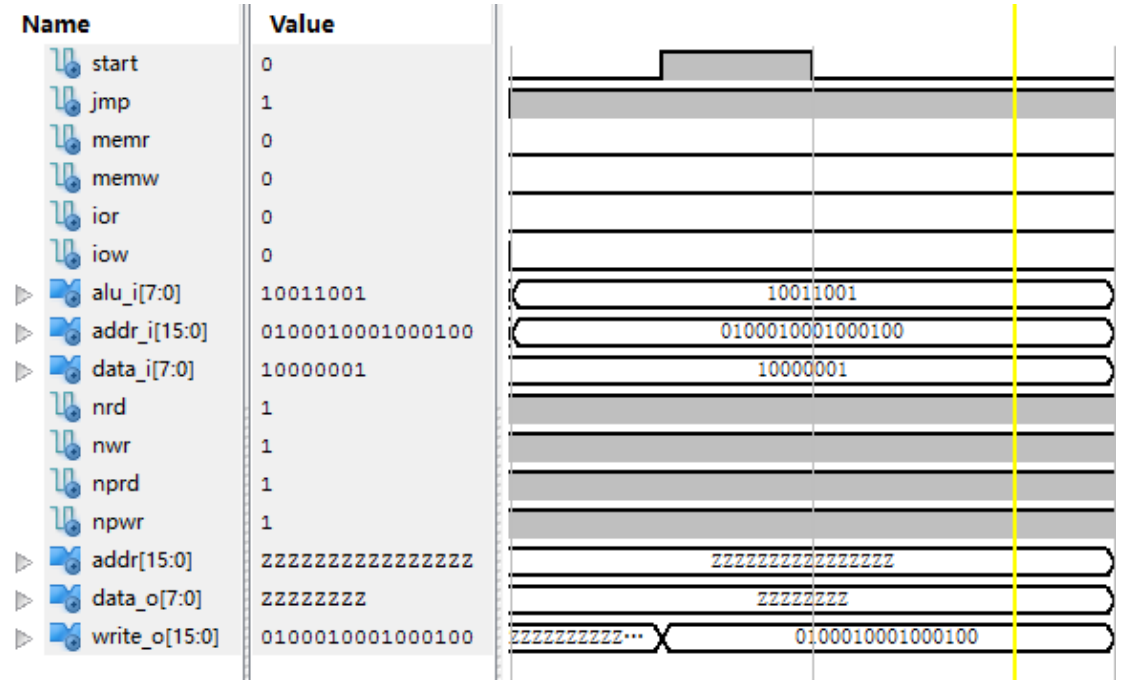
IN：

由仿真波形可知在 IN 指令周期内，在访存周期时外设读请求 nprd 低电平有效，同时 addr 输出来自 addr_i 的地址 1111000011110000，write_o 低八位输出来自 data_i 的数据输入 10000001，高八位置为“00000000”。



OUT：

由仿真波形可知在 OUT 指令周期内，在访存周期时外设写请求 npwr 低电平有效，同时 addr 输出来自 addr_i 的地址 0011001100110011，data_o 输出来自 alu_i 的数据 10101010，由于无需回写，write_o 输出信号被置为高阻态。



JMP：

由仿真波形可知在 JMP 指令周期内，在访存周期无读写请求发出，addr 与 data_o 均置为高阻态，write_o 输出来自 addr_i 的地址输入 0100010001000100 以便在回写周期更新 PC。

5.回写管理模块

5.1 测试方案

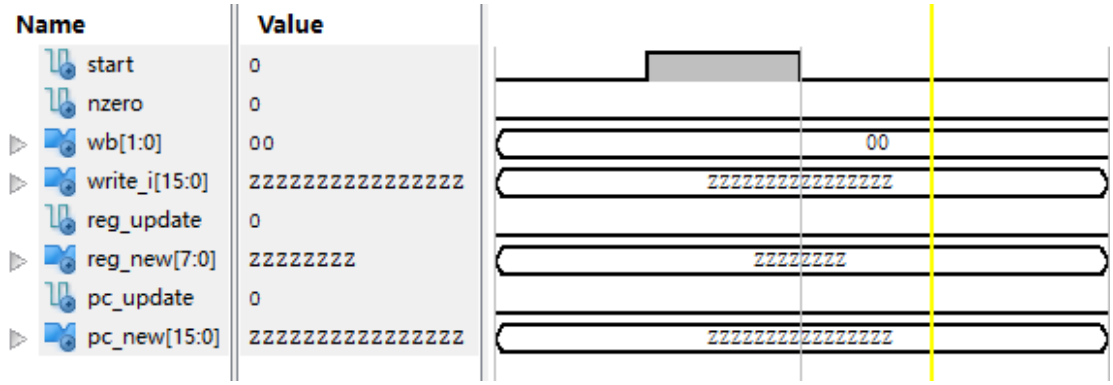
①令回写方式控制信号 wb 为“00”，观察各输出信号。
②令回写方式控制信号 wb 为“01”，观察各输出信号。
③令回写方式控制信号 wb 为“10”，nzero 为低电平，观察各输出信号。
④令回写方式控制信号 wb 为“10”，nzero 为高电平，观察各输出信号。
⑤令回写方式控制信号 wb 为“11”，观察各输出信号。

5.2 测试过程

```
stim_proc: process
   begin
        start <= '0';
        write_i <= (others=>'Z');
      wait for 100 ns;

        start <= '1';
        wb <= "00";
        nzero <= '0';
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        write_i <= "1111111100000000";
        wait for 100 ns;

        start <= '1';
        wb <= "01";
        nzero <= '0';
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        write_i <= "0000111100001111";
        wait for 100 ns;

        start <= '1';
        wb <= "10";
        nzero <= '0';
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        write_i <= "1111000011110000";
        wait for 100 ns;

        start <= '1';
        wb <= "10";
        nzero <= '1';
```

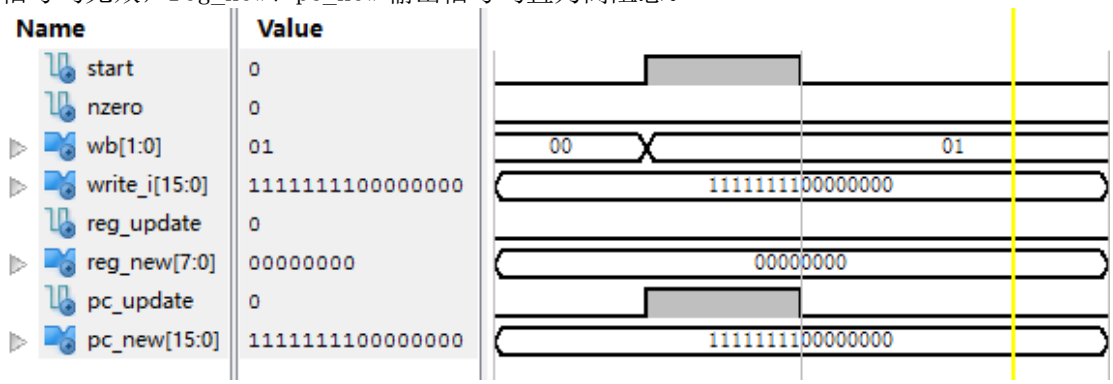
```
        wait for 100 ns;

        start <= '0';
        wait for 200 ns;

        write_i <= "0000000011001100";
        wait for 100 ns;

        start <= '1';
        wb <= "11";
        nzero <= '0';
        wait for 100 ns;

        start <= '0';

      wait;
   end process;
```

5.3 仿真波形



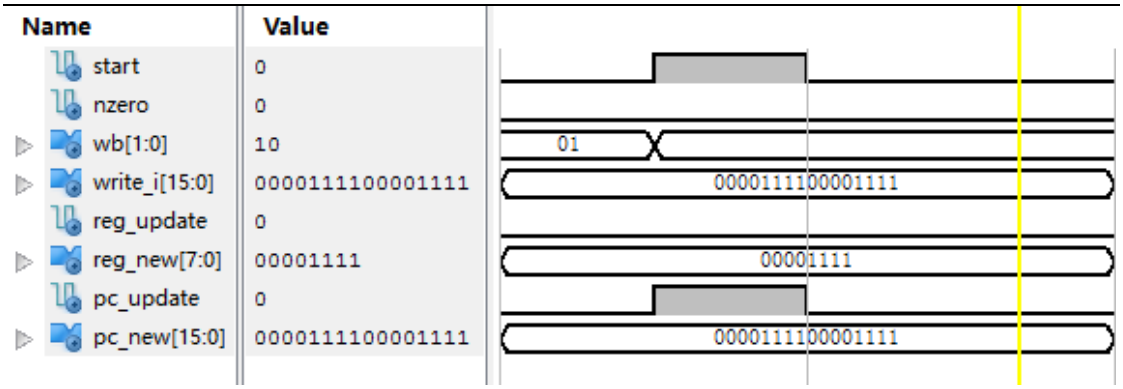
wb = “00”：

由仿真波形可知回写方式为“00”时，无需回写，因此在访存周期 reg_update、pc_update 信号均无效，reg_new、pc_new 输出信号均置为高阻态。



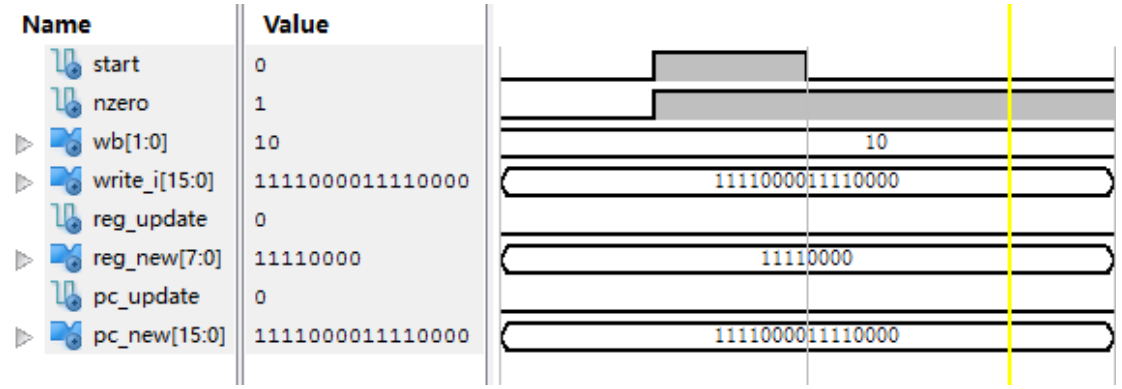
wb = “01”：

由仿真波形可知回写方式为“01”时，直接回写到 PC，因此在访存周期 pc_update 信号有效，pc_new 输出 write_i 输入信号的内容。



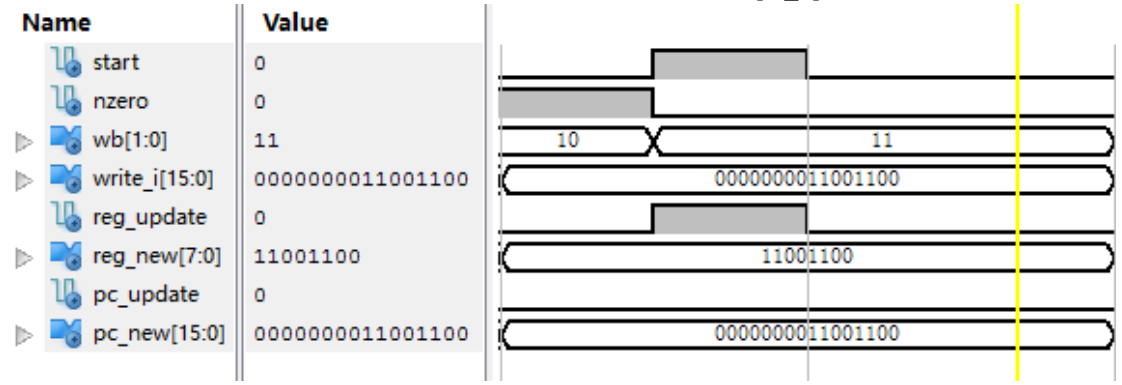
wb = “10”, nzero = ‘0’:

由仿真波形可知回写方式为“10”时，需要判断是否回写到 PC，由于 nzero 信号为低电平，因此在访存周期 pc_update 信号有效，pc_new 输出 write_i 输入信号的内容。



wb = “10”, nzero = ‘1’:

由仿真波形可知回写方式为“10”时，需要判断是否回写到 PC，由于 nzero 信号为高电电平，表示寄存器内容不是“00000000”，因此在访存周期 pc_update 信号无效。



wb = “11”:

由仿真波形可知回写方式为“11”时，需要回写到通用寄存器，因此在访存周期 reg_update 信号有效，reg_new 输出 write_i 输入信号低八位内容。

6.访存控制模块

6.1 测试方案

①模拟 MVI 指令的指令周期控制信号，观察各输出信号。
②模拟 OUT 指令的指令周期控制信号，观察各输出信号。
③模拟 STA 指令的指令周期控制信号，观察各输出信号。

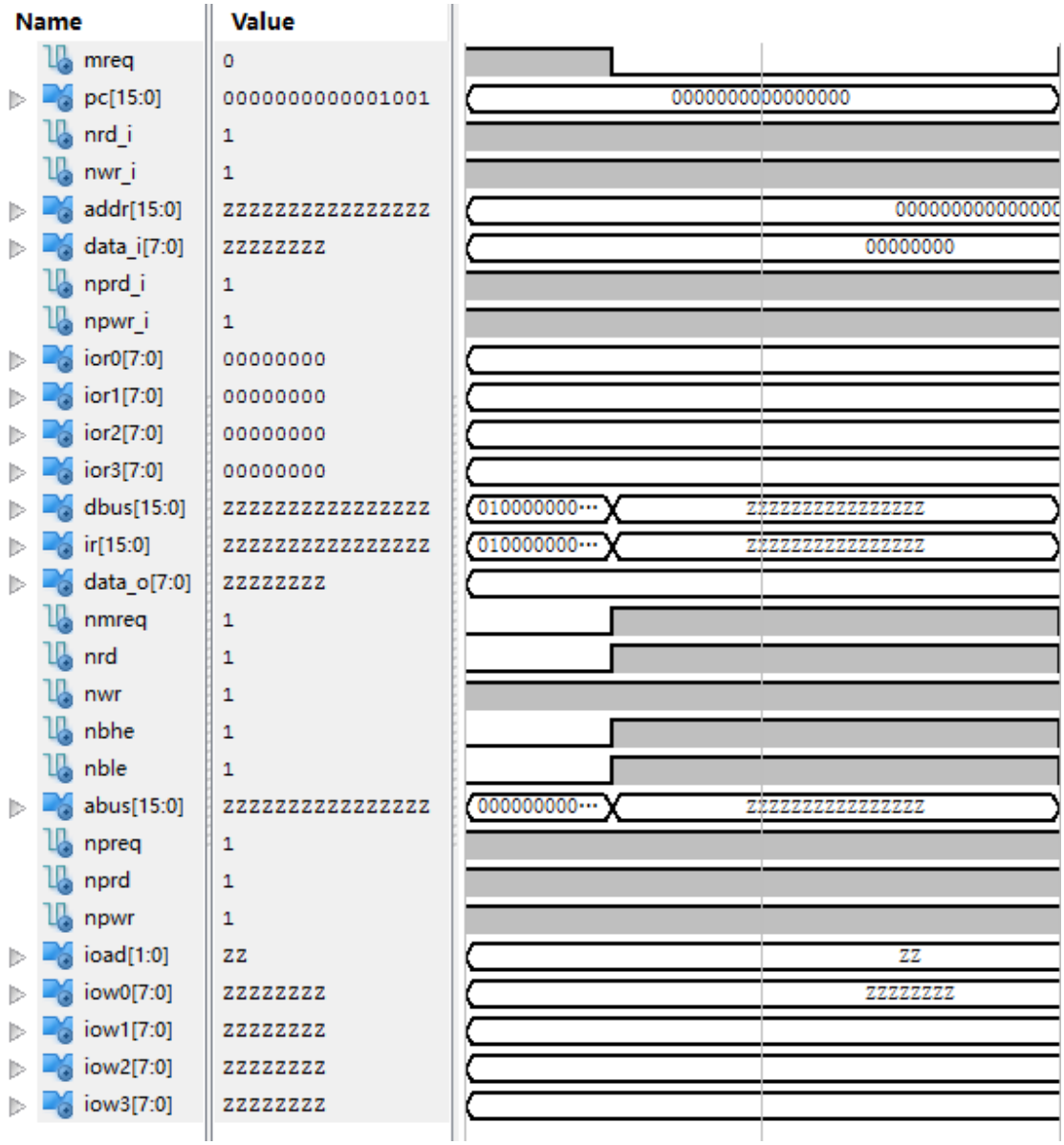
④模拟 STA 指令的指令周期控制信号，观察各输出信号。

6.2 测试过程

```
stim_proc: process
   begin
        mreq <= '1';
        pc <= "0000000000000000";
        DBus <= "0100000000000000";      --MVI
        wait for 100 ns;

        mreq <= '0';
        DBus <= (others=>'Z');
        wait for 300 ns;

        mreq <= '1';
        pc <= "0000000000000001";
        DBus <= "1000000000000000";      --OUT
        wait for 100 ns;

        mreq <= '0';
        DBus <= (others=>'Z');
        wait for 100 ns;

        npwr_i <= '0';
        addr <= "0000000000000000";
        data_i <= "00110011";
        wait for 100 ns;

        npwr_i <= '1';
        addr <= (others=>'Z');
        data_i <= (others=>'Z');
        wait for 100 ns;

        mreq <= '1';
        pc <= "0000000000001000";
        DBus <= "0110001000000000";      --STA
        wait for 100 ns;

        mreq <= '0';
        DBus <= (others=>'Z');
        wait for 100 ns;

        nwr_i <= '0';
        addr <= "0000000100000000";
        data_i <= "11001100";
        wait for 100 ns;

        nwr_i <= '1';
        addr <= (others=>'Z');
        data_i <= (others=>'Z');
        wait for 100 ns;

        mreq <= '1';
        pc <= "0000000000001001";
        DBus <= "0111001100000000";      --LDA
```

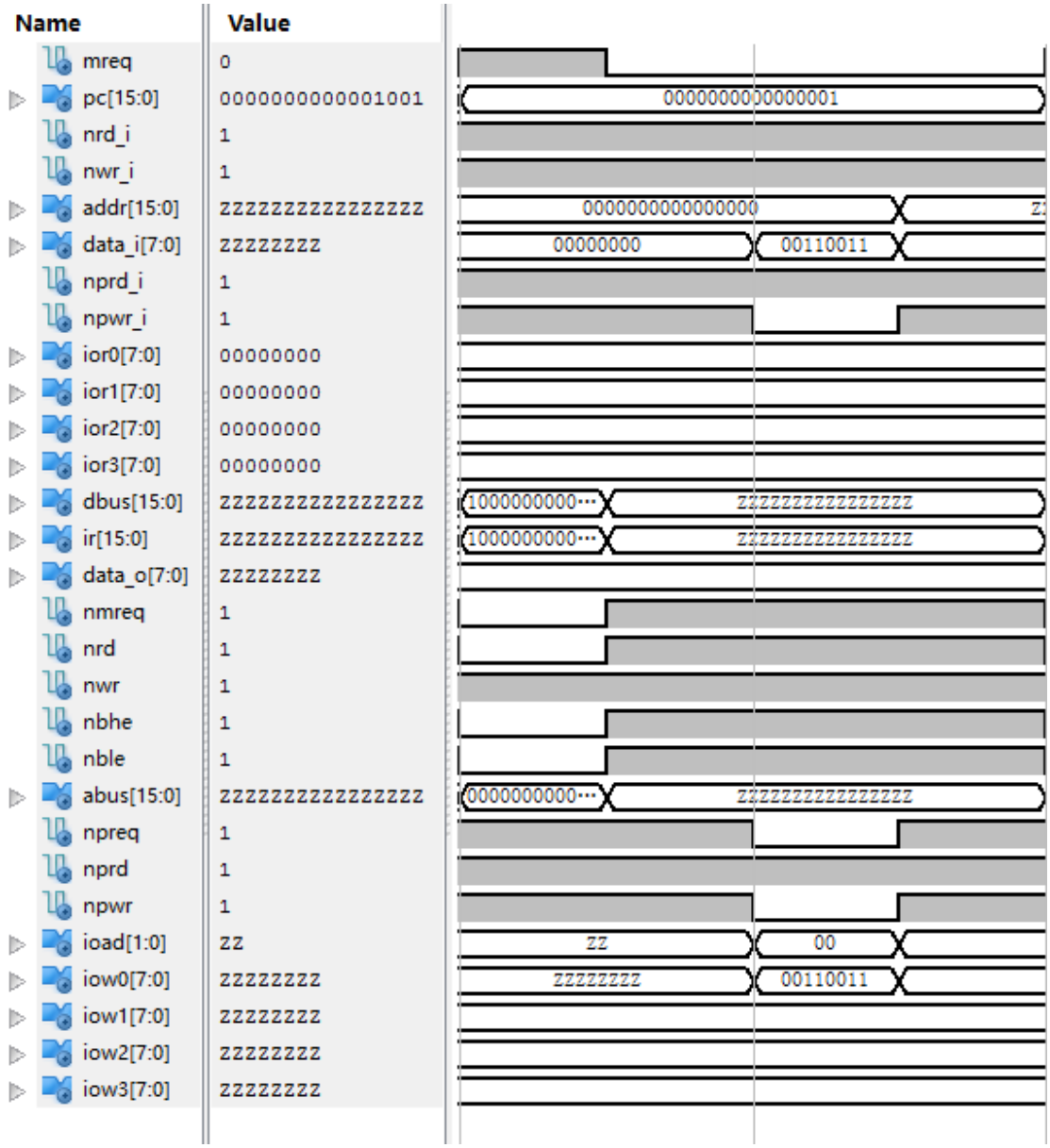
```
        wait for 100 ns;

        mreq <= '0';
        DBus <= (others=>'Z');
        wait for 100 ns;

        nrd_i <= '0';
        addr <= "0000000110000001";
        DBus <= "1010101001010101";
        wait for 100 ns;

        nrd_i <= '1';
        DBus <= (others=>'Z');
        addr <= (others=>'Z');

      wait;
   end process;
```

6.3 仿真波形



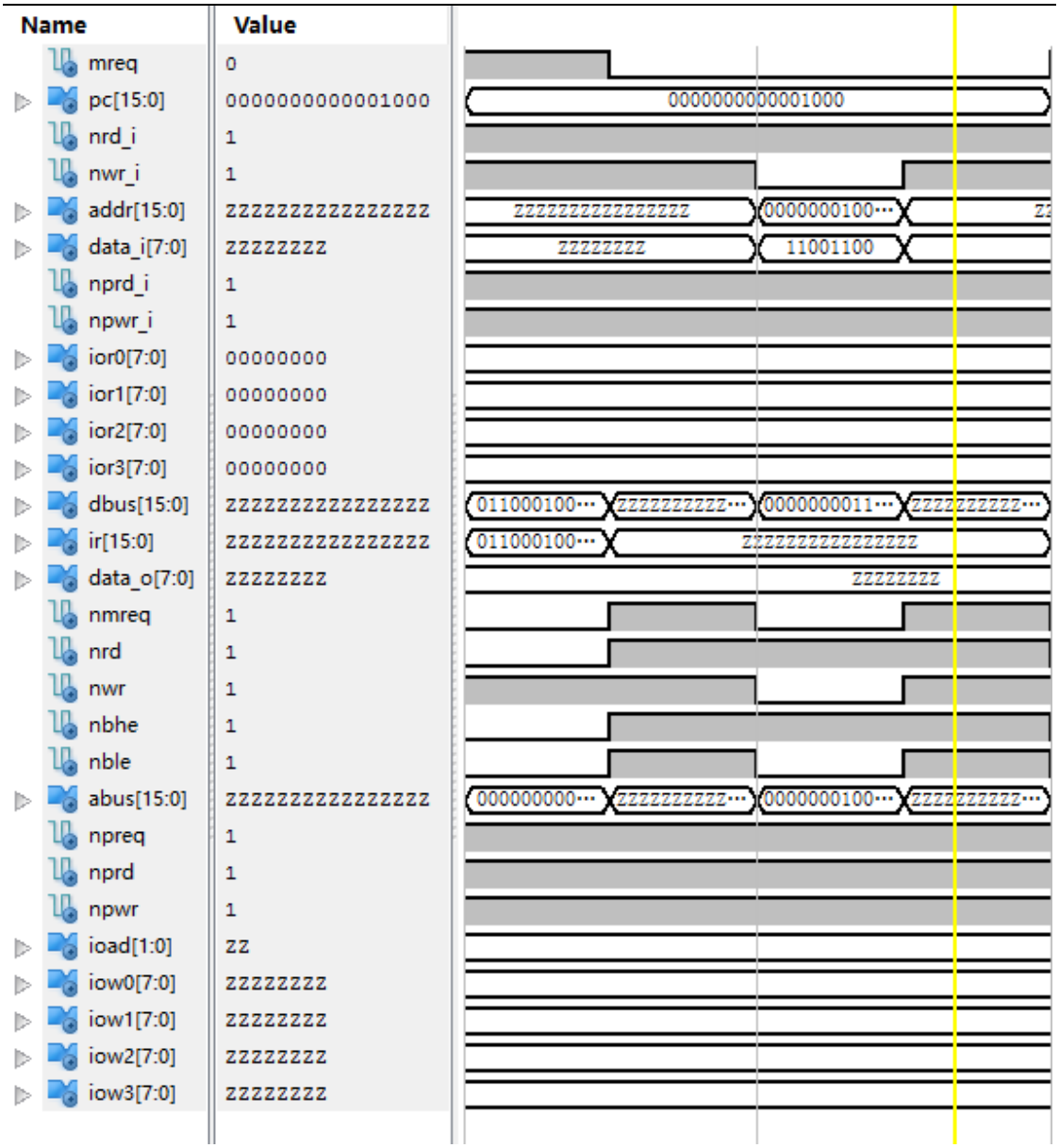
STA:

由仿真波形可知在取指周期，nmreq，nrd，nbhe，nble 为低电平有效，完成取指令操作。由于 MVI 指令无需访存，因此在访存周期无操作。



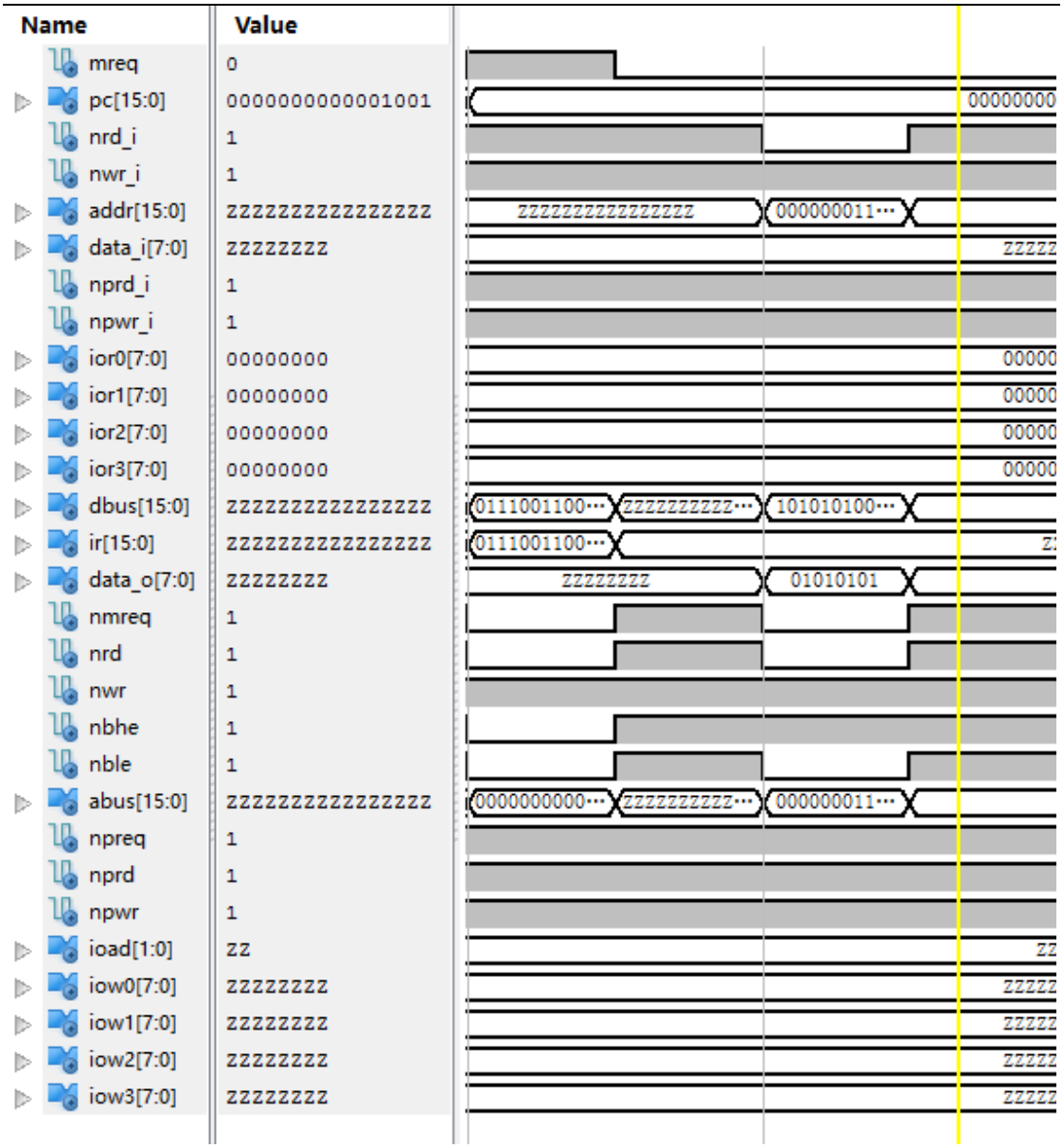
OUT:

由仿真波形可知在取指周期，nmreq，nrd，nbhe，nble 为低电平有效，完成取指令操作。在访存周期，控制外设的 npreq 和 npwr 信号为低电平有效，IOAD 外设地址输出为 addr 的低两位“00”，输出端口 IOW0 输出来自 data_i 的数据输入。



STA:

由仿真波形可知在取指周期，nmreq，nrd，nbhe，nble 为低电平有效，完成取指令操作。在访存周期，控制主存的 nmreq，nwr，nble 信号为低电平有效，地址总线 ABus 输出来自 addr 的地址，数据总线的低八位输出来自 data_i 的数据 11001100，完成主存写操作。



LDA：

由仿真波形可知在取指周期，nmreq，nrd，nbhe，nble 为低电平有效，完成取指令操作。在访存周期，控制主存的 nmreq，nrd，nble 信号为低电平有效，地址总线 ABus 输出来自 addr 的地址，将数据总线 DBus 上读出的 16 位数据 1010101001010101 的低八位 01010101 输出到 data_o，完成主存读操作。

7.CPU 顶层模块

7.1 测试方案

(1)令系统复位信号 RST 有效，并保持 40ns，使取指管理模块的 PC 信号获得初始值“0000000000000000”。

(2)通过数据总线 DBus 输入指令“0100011100000001”（MVI R7, 00000001），观察指令周期内输出信号的变化情况。

(3)通过数据总线 DBus 输入指令“1000011100000011”（OUT R7, IOW3），观察指令周期内输

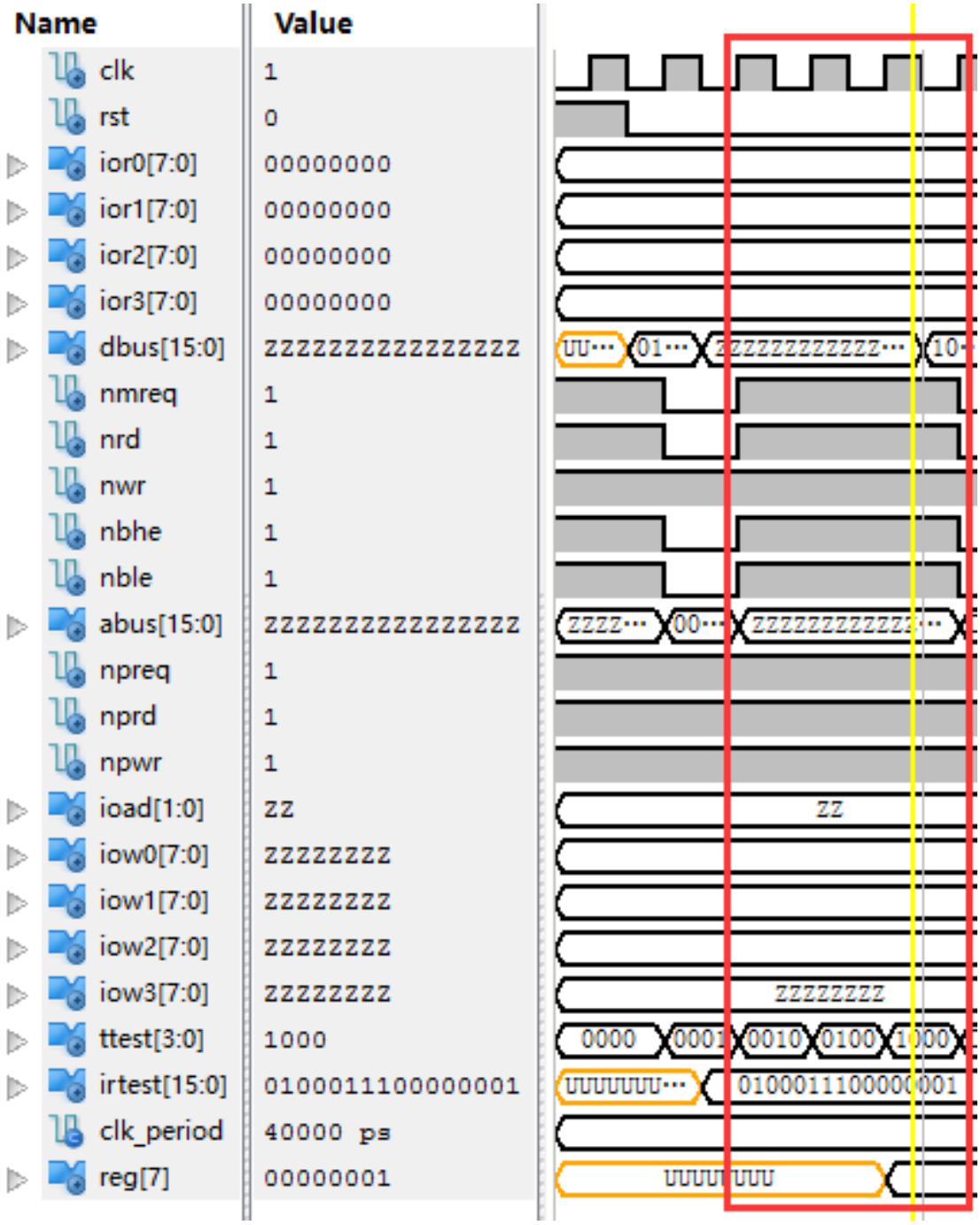
出信号的变化情况。
(4)通过数据总线 DBus 输入指令“0100000100001110”（MVI R1, 00001110），观察指令周期内输出信号的变化情况。
(5)通过数据总线 DBus 输入指令“0010000100000111”（SUB R1, R7），观察指令周期内输出信号的变化情况。
(6)通过数据总线 DBus 输入指令“0101000100000111”（MOV R1, R7），观察指令周期内输出信号的变化情况。
(7)通过数据总线 DBus 输入指令“0011000100000111”（ADD R1, R7），观察指令周期内输出信号的变化情况。
(8)通过数据总线 DBus 输入指令“0110000100000000”（STA R1, 0000000100000000），观察指令周期内输出信号的变化情况。
(9)通过数据总线 DBus 输入指令“0111001000000000”（LDA R2, 0000000100000000），观察指令周期内输出信号的变化情况。
(10)通过数据总线 DBus 输入指令“1000000100000001”（OUT R1, IOW1），观察指令周期内输出信号的变化情况。
(11)通过数据总线 DBus 输入指令“1001011100000010”（IN R7, IOR2），观察指令周期内输出信号的变化情况。
(12)通过数据总线 DBus 输入指令“0000000000000000”（JMP 0011001100000000），观察指令周期内输出信号的变化情况。
(13)通过数据总线 DBus 输入指令“0001000000001111”（JZ 0011001100001111），观察指令周期内输出信号的变化情况。

7.2 测试过程

```
stim_proc: process
   begin
      rst <= '1';
        wait for 40 ns;
        rst <= '0';

        DBus <= "0100011100000001";  --MVI R7, 00000001
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "1000011100000011";  --OUT R7, IOW3
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0100000100001110";  --MVI R1, 00001110
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0010000100000111";  --SUB R1, R7
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0101000100000111";  --MOV R1, R7
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;
```

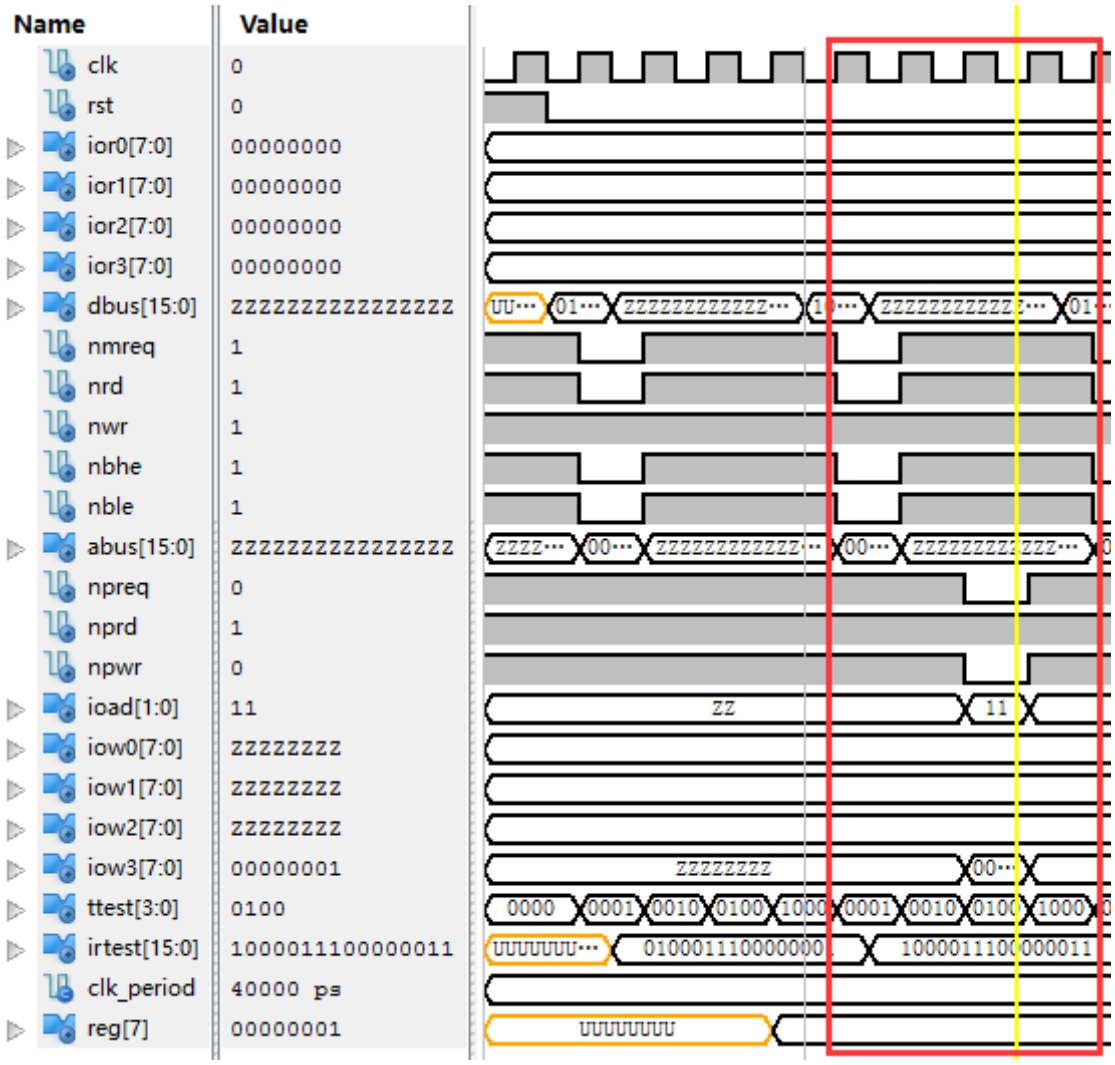
```
        DBus <= "0011000100000111";  --ADD R1, R7
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0110000100000000";  --STA R1, 0000000100000000
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0111001000000000";  --LDA R2, 0000000100000000
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 40 ns;
        DBus <= "0000000000000010";
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 40 ns;

        DBus <= "1000000100000001";  --OUT R1, IOW1
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "1001011100000010";  --IN R7, IOR2
        wait for 40 ns;
        DBUs <= (others=>'Z');
        wait for 40 ns;
        IOR2 <= "00110011";
        wait for 80 ns;

        DBus <= "0000000000000000";  --JMP 0011001100000000
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

        DBus <= "0001000000001111";  --JZ 0011001100001111
        wait for 40 ns;
        DBus <= (others=>'Z');
        wait for 120 ns;

      wait;
    end process;
```

7.3 仿真波形



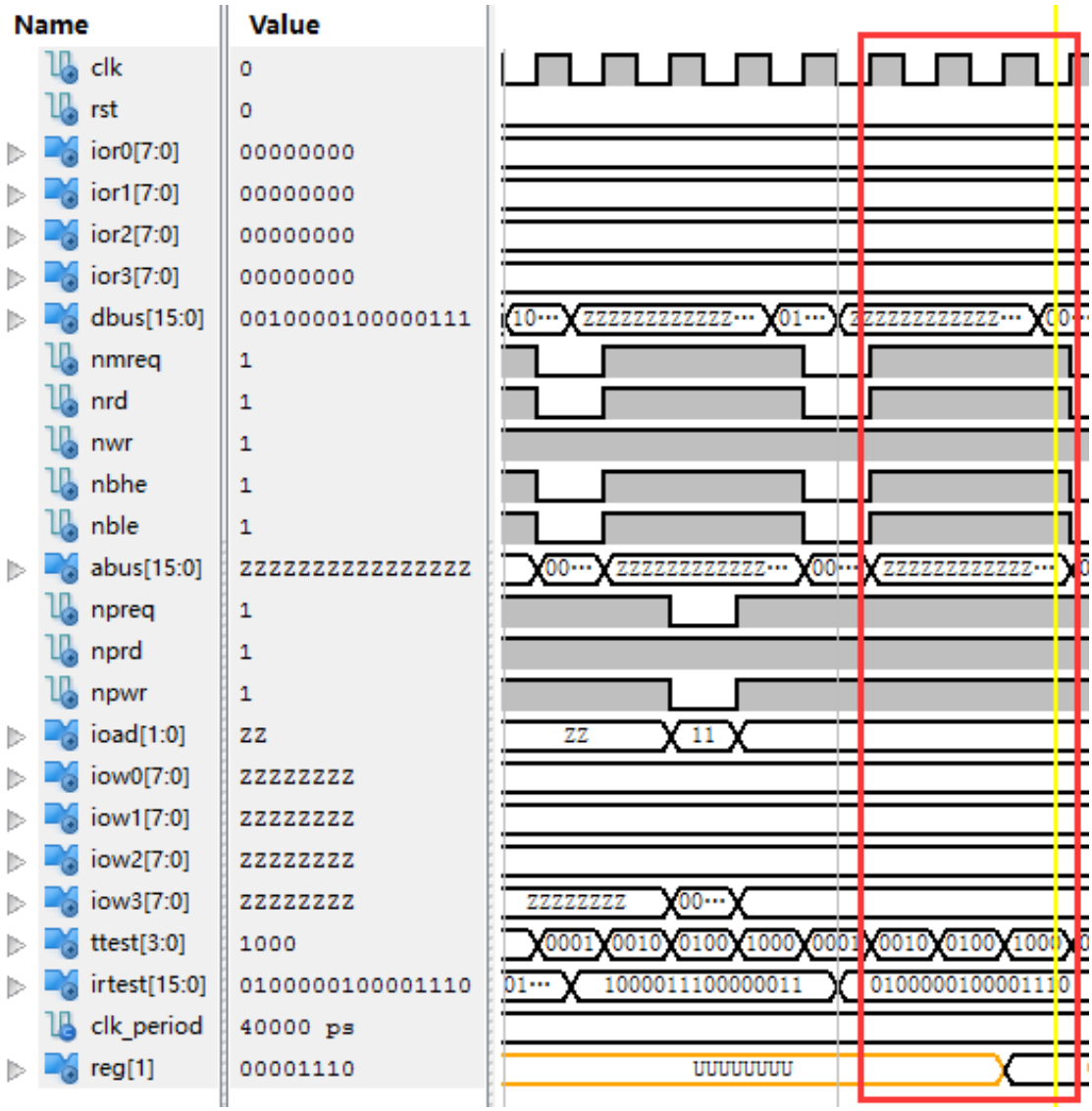
MVI R7, 00000001:

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 MVI 指令；在访存周期内，无读写信号产生；在回写周期内，运算管理模块的通用寄存器 reg(7)成功更新为立即数 00000001。



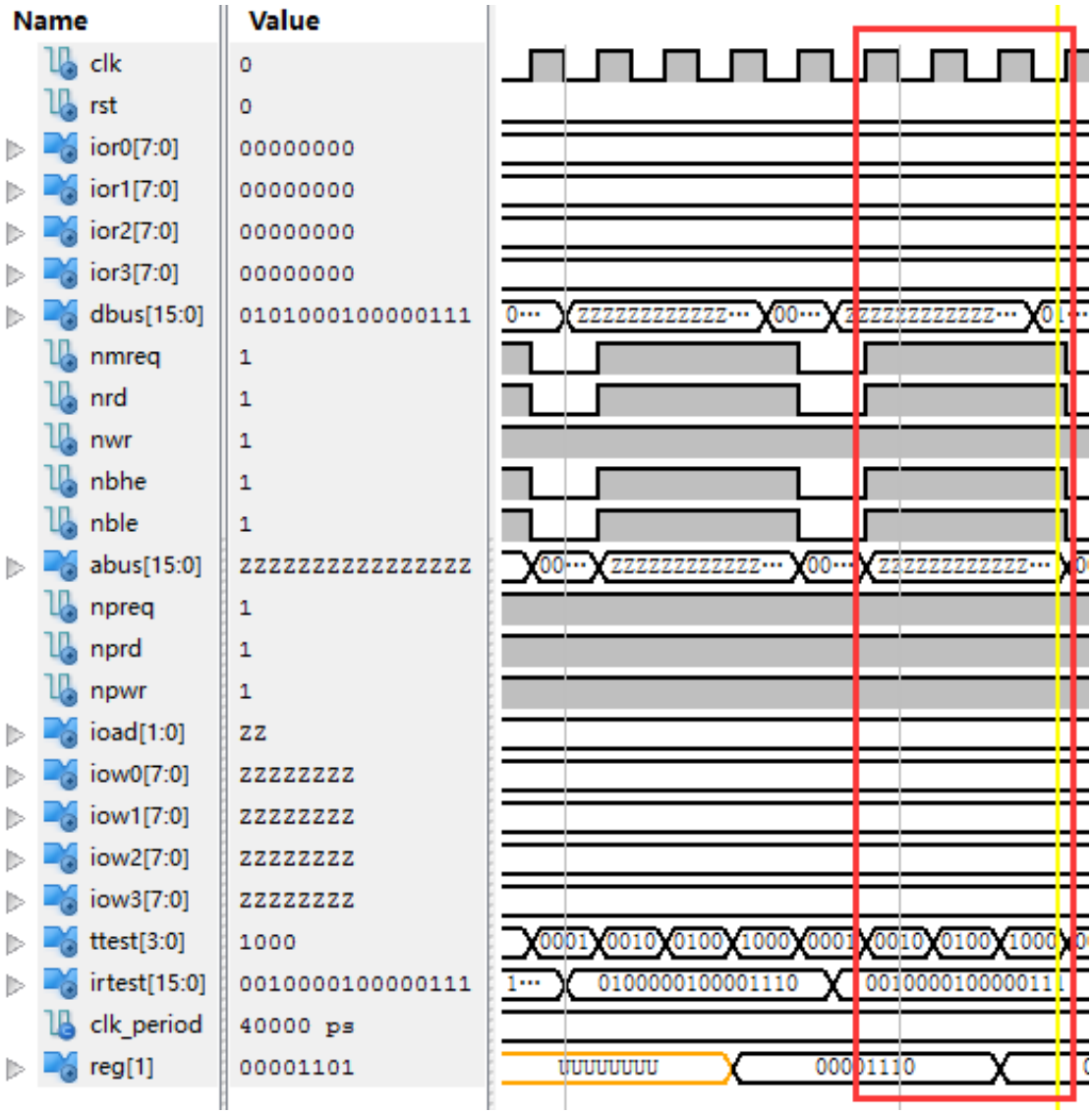
OUT R7, IOW3：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 OUT 指令；在访存周期内，外设片选信号 npreq 和外设写信号 npwr 有效，IOAD 输出为“11”，reg(7)中的数据 00000001 被输出到 IOW3 端口。



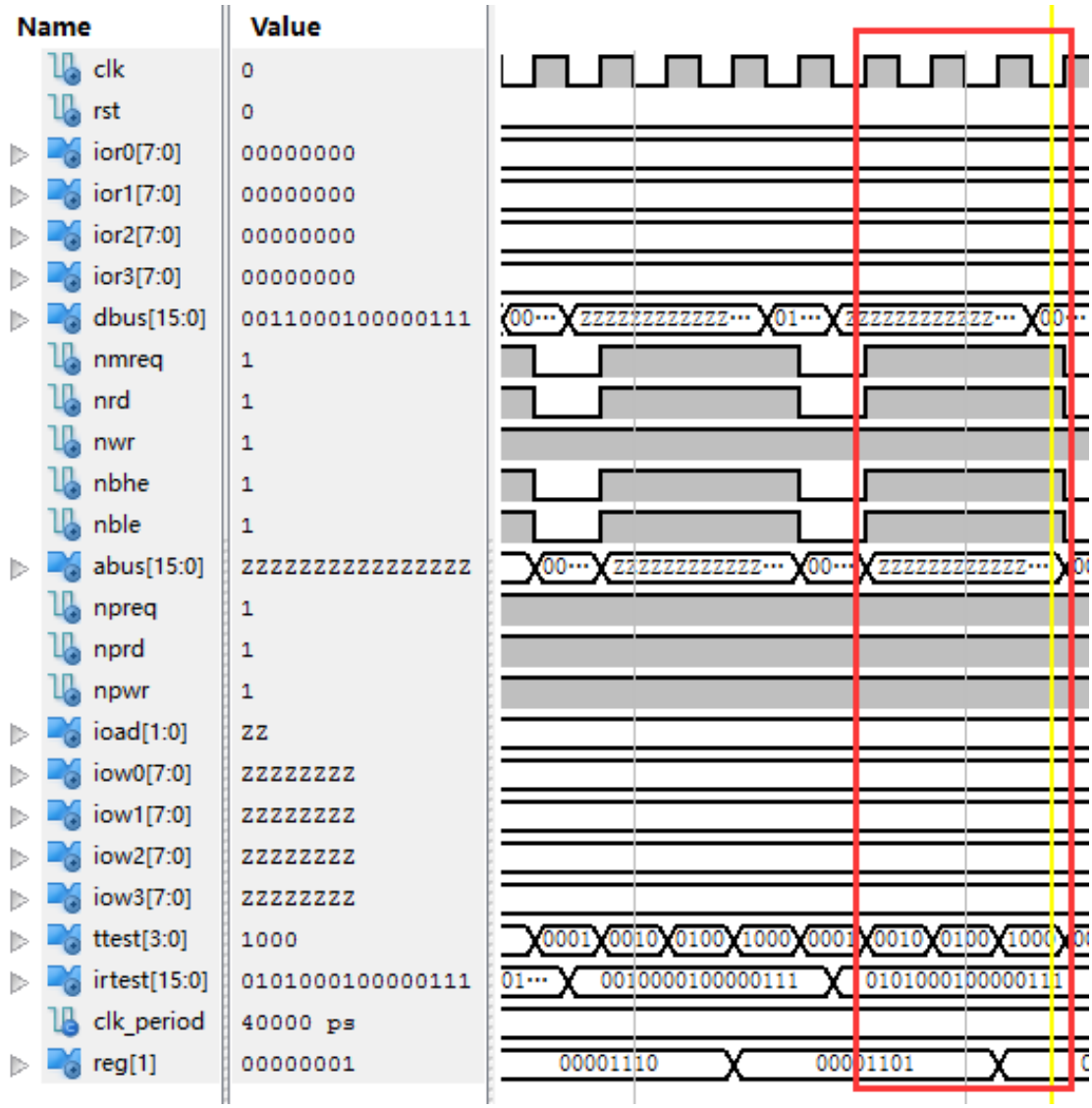
MVI R1, 00001110:

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 MVI 指令；在访存周期内，无读写信号产生；在回写周期内，运算管理模块的通用寄存器 reg(1)成功更新为立即数 00001110。



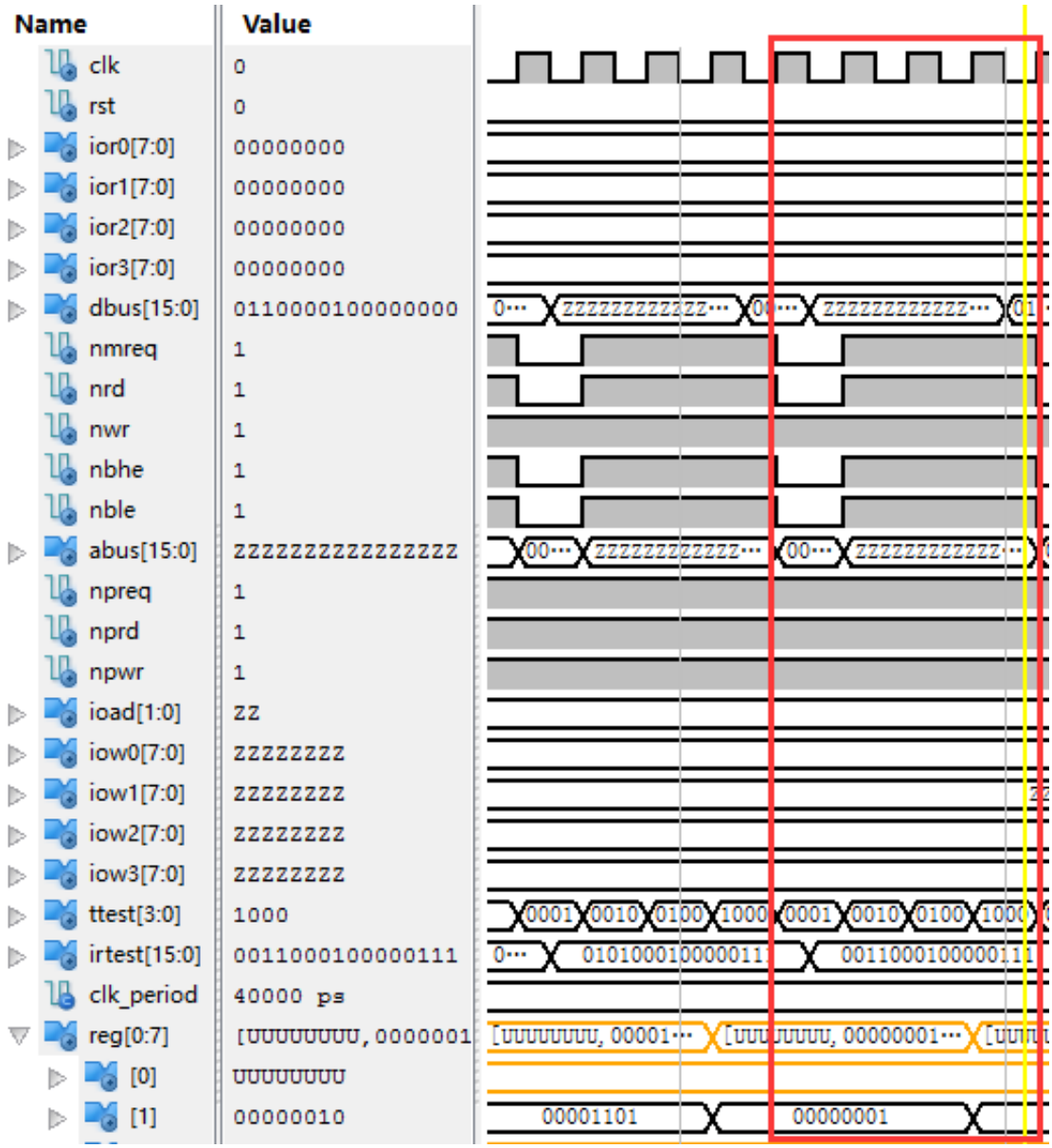
SUB R1, R7:

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 SUB 指令；在访存周期内，无读写信号产生；在回写周期内，运算管理模块的通用寄存器 reg(1)成功更新为 reg(1)-reg(7)的结果 00001101。



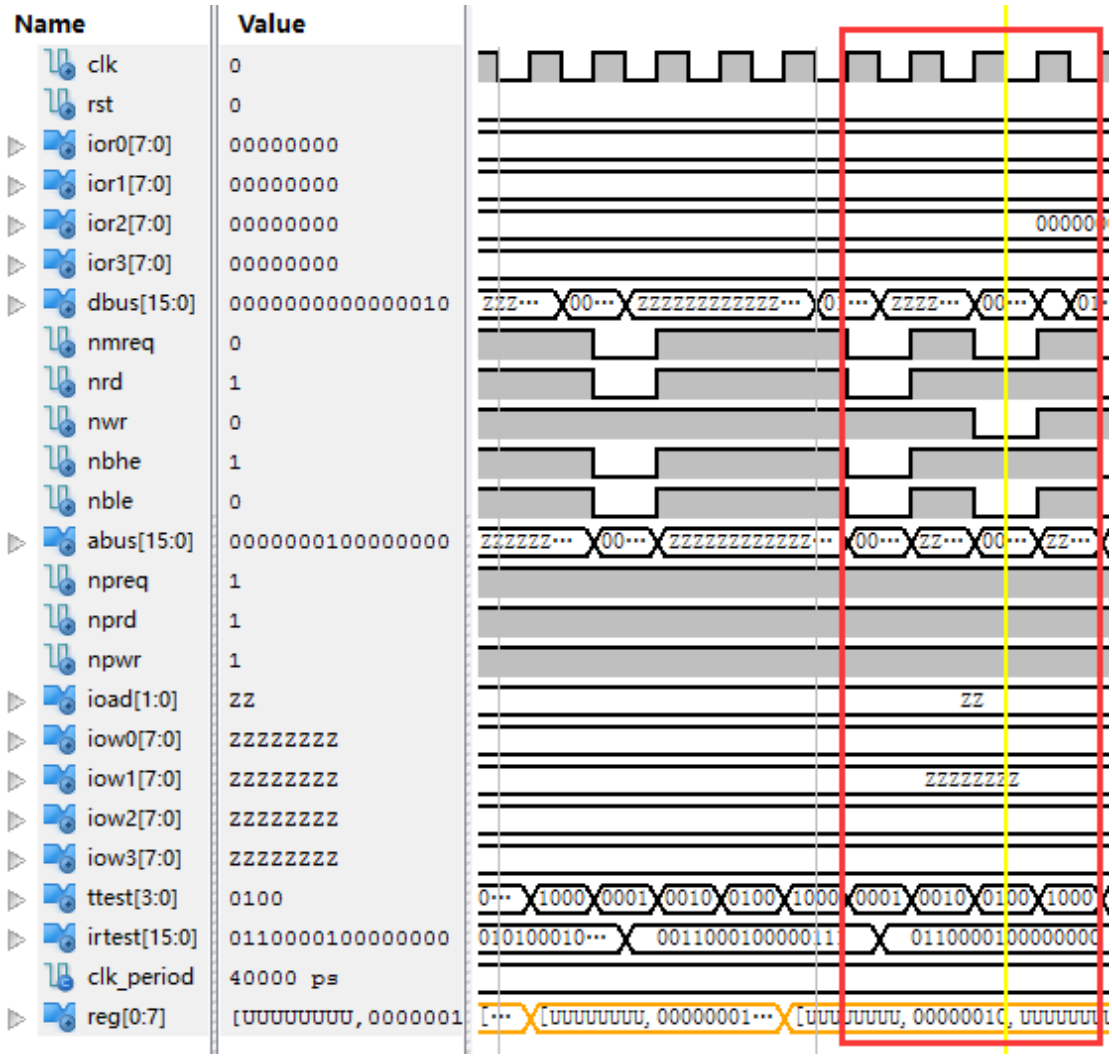
MOV R1, R7：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 MOV 指令；在访存周期内，无读写信号产生；在回写周期内，运算管理模块的通用寄存器 reg(1)成功更新为 reg(7)的值 00000001。



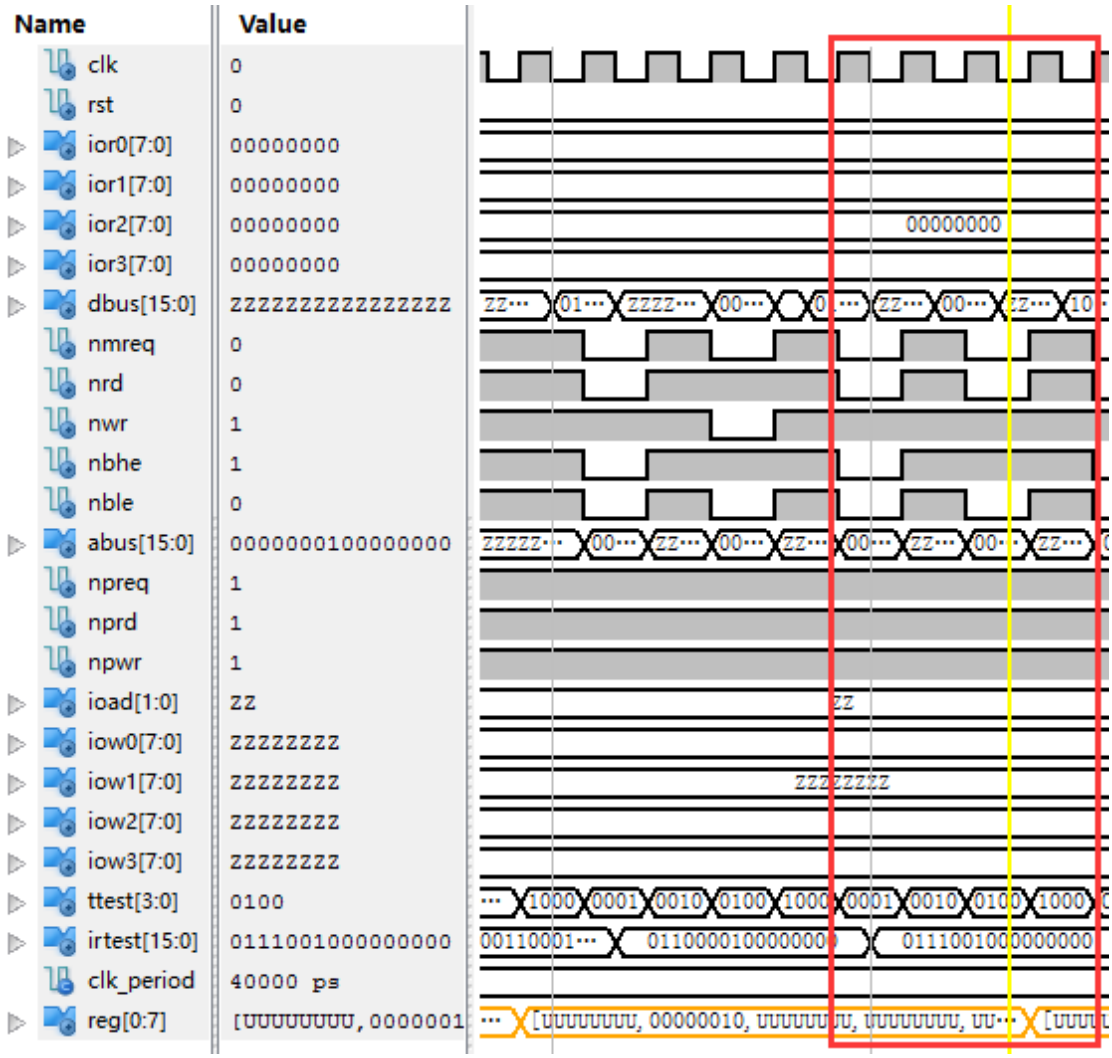
ADD R1, R7：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 ADD 指令；在访存周期内，无读写信号产生；在回写周期内，运算管理模块的通用寄存器 reg(1)成功更新为 reg(1)+reg(7)的结果 000000010。



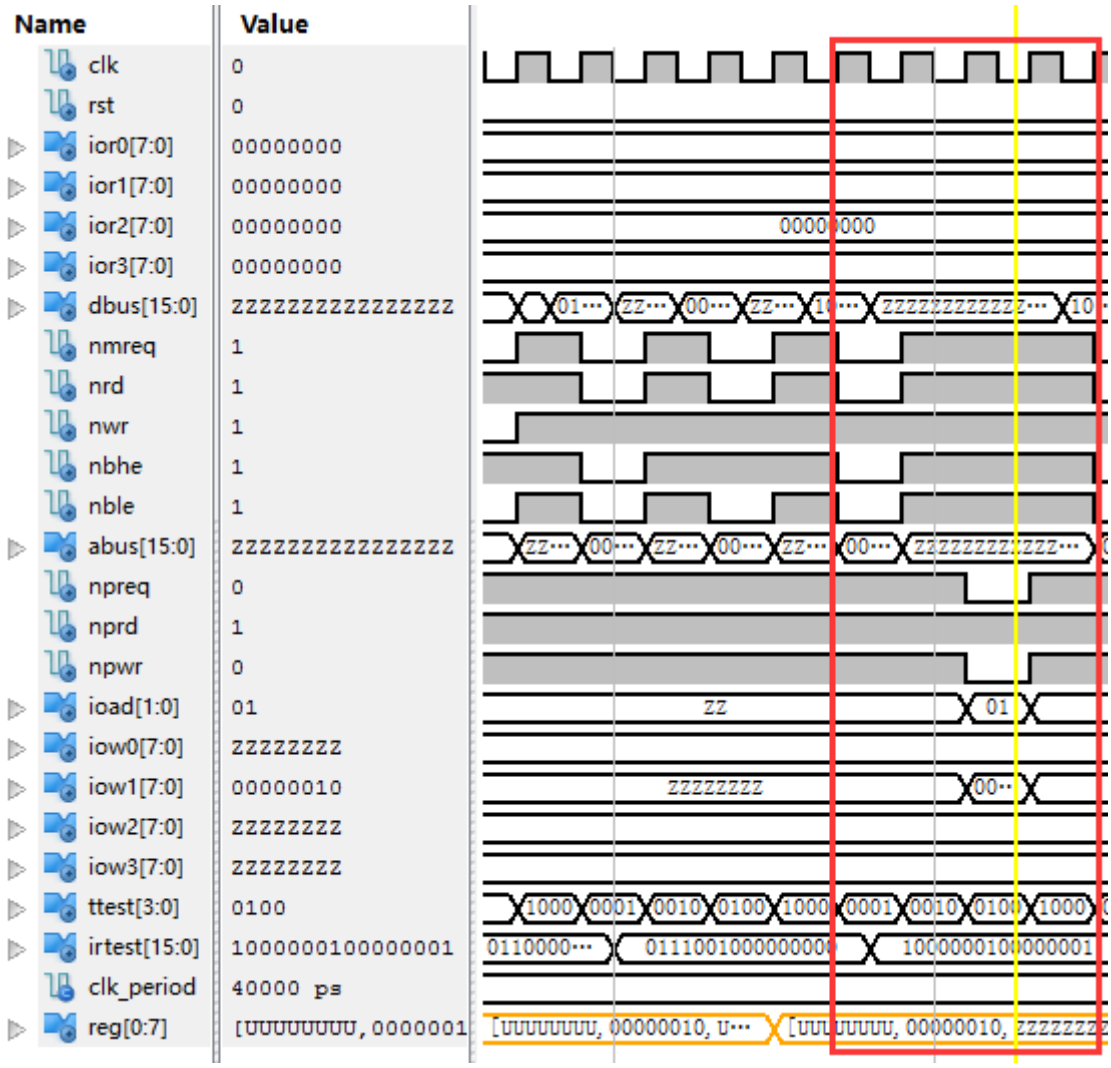
STA R1, 0000000100000000：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 STA 指令；在访存周期内，存储器片选信号 nmreq，存储器写信号 nwr，低字节访问允许信号 nble 有效，地址总线 ABus 输出访存地址，数据总线 DBus 低八位输出 reg(1)寄存器内容，完成写入存储器操作。



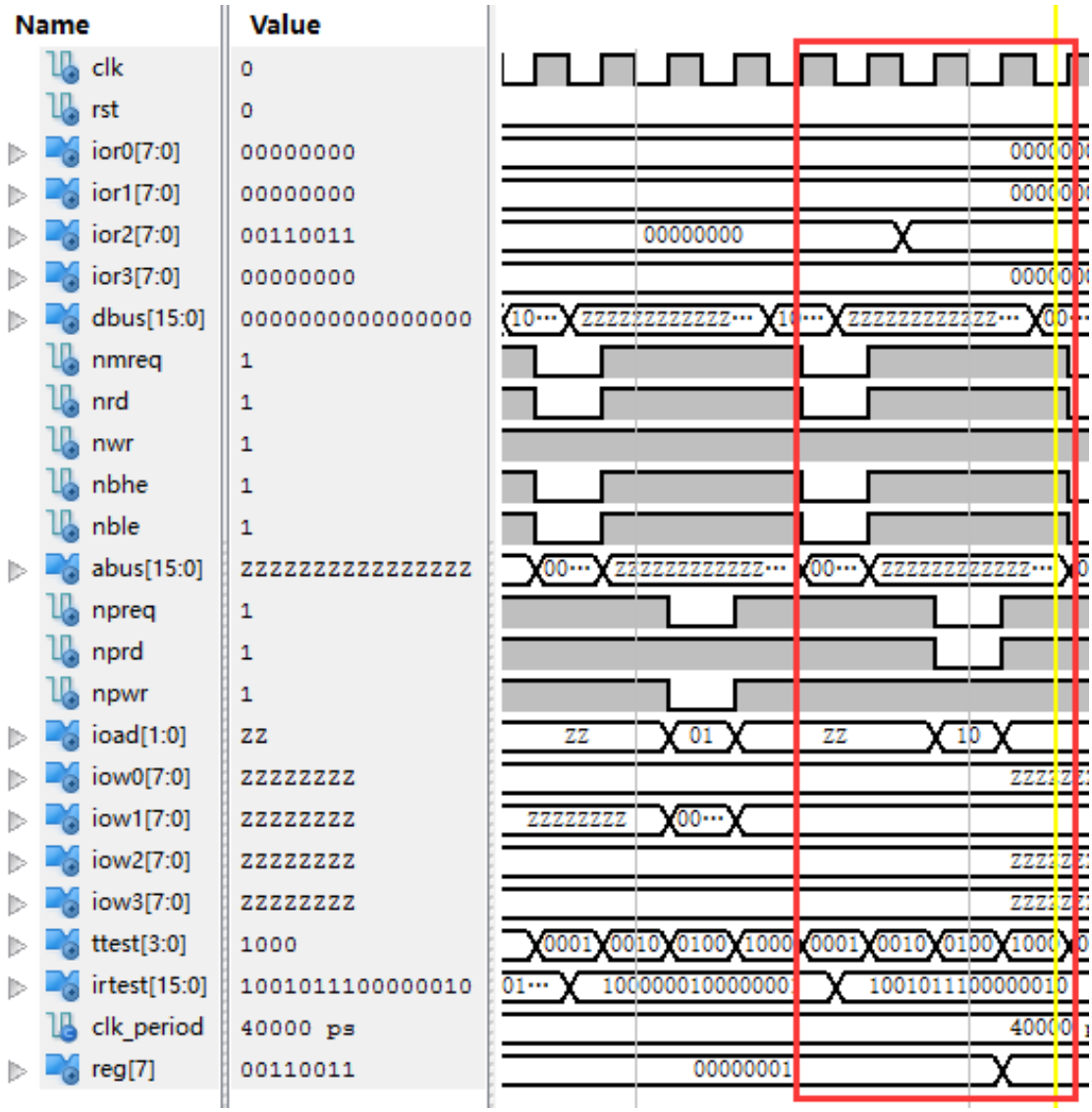
LDA R2, 0000000100000000：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 LDA 指令；在访存周期内，存储器片选信号 nmreq，存储器读信号 nrd，低字节访问允许信号 nble 有效，地址总线 ABus 输出访存地址，完成读取存储器操作。



OUT R1, IOW1：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 OUT 指令；在访存周期内，外设片选信号 npreq 和外设写信号 npwr 有效，IOAD 输出为“01”，reg(1)中的数据 00000010 被输出到 IOW1 端口。



IN R7, IOR2：

由仿真波形可知，在取指周期内，处理器发出相应的访存信号取出 IN 指令；在访存周期内，外设片选信号 npreq 和外设读信号 nprd 有效，IOAD 输出为“10”；在回写周期，IOR2 端口的数据 00110011 被写入 reg(7)寄存器中。

五．管脚定义

```
NET "CLK"  LOC = "p75"  ;
NET "RST"  LOC = "p74";

NET "ABus<0>"  LOC = "p179";
NET "ABus<1>"  LOC = "p178";
NET "ABus<2>"  LOC = "p177";
NET "ABus<3>"  LOC = "p172";
NET "ABus<4>"  LOC = "p171";
NET "ABus<5>"  LOC = "p151";
NET "ABus<6>"  LOC = "p150";
NET "ABus<7>"  LOC = "p147";
NET "ABus<8>"  LOC = "p146"  ;
NET "ABus<9>"  LOC = "p113"  ;
NET "ABus<10>"  LOC = "p115"  ;
NET "ABus<11>"  LOC = "p116"  ;
NET "ABus<12>"  LOC = "p119"  ;
NET "ABus<13>"  LOC = "p140"  ;
NET "ABus<14>"  LOC = "p144"  ;
NET "ABus<15>"  LOC = "p145"  ;

NET "DBus<0>"  LOC = "p167"  ;
NET "DBus<1>"  LOC = "p165"  ;
NET "DBus<2>"  LOC = "p164"  ;
NET "DBus<3>"  LOC = "p163"  ;
NET "DBus<4>"  LOC = "p162"  ;
NET "DBus<5>"  LOC = "p161"  ;
NET "DBus<6>"  LOC = "p160"  ;
NET "DBus<7>"  LOC = "p153"  ;
NET "DBus<8>"  LOC = "p120"  ;
NET "DBus<9>"  LOC = "p122"  ;
NET "DBus<10>"  LOC = "p123"  ;
NET "DBus<11>"  LOC = "p128"  ;
NET "DBus<12>"  LOC = "p132"  ;
NET "DBus<13>"  LOC = "p133"  ;
NET "DBus<14>"  LOC = "p134"  ;
NET "DBus<15>"  LOC = "p135"  ;

NET "IOR0<0>"  LOC = "p154"  ;
NET "IOR0<1>"  LOC = "p148"  ;
NET "IOR0<2>"  LOC = "p142"  ;
NET "IOR0<3>"  LOC = "p136"  ;
NET "IOR0<4>"  LOC = "p130"  ;
NET "IOR0<5>"  LOC = "p124"  ;
NET "IOR0<6>"  LOC = "p118"  ;
NET "IOR0<7>"  LOC = "p110"  ;

NET "IOR1<0>"  LOC = "p174"  ;
NET "IOR1<1>"  LOC = "p204"  ;
NET "IOR1<2>"  LOC = "p194"  ;
NET "IOR1<3>"  LOC = "p175"  ;
```

```
NET "IOR1<4>"  LOC = "p169"  ;
NET "IOR1<5>"  LOC = "p101"  ;
NET "IOR1<6>"  LOC = "p97"  ;
NET "IOR1<7>"  LOC = "p96"  ;

NET "IOR2<0>"  LOC = "p94"  ;
NET "IOR2<1>"  LOC = "p93"  ;
NET "IOR2<2>"  LOC = "p91"  ;
NET "IOR2<3>"  LOC = "p90"  ;
NET "IOR2<4>"  LOC = "p89"  ;
NET "IOR2<5>"  LOC = "p87"  ;
NET "IOR2<6>"  LOC = "p80"  ;
NET "IOR2<7>"  LOC = "p78"  ;

NET "IOR3<0>"  LOC = "p72"  ;
NET "IOR3<1>"  LOC = "p71"  ;
NET "IOR3<2>"  LOC = "p69"  ;
NET "IOR3<3>"  LOC = "p68"  ;
NET "IOR3<4>"  LOC = "p65"  ;
NET "IOR3<5>"  LOC = "p64"  ;
NET "IOR3<6>"  LOC = "p58"  ;
NET "IOR3<7>"  LOC = "p57"  ;


NET "IOW0<0>"  LOC = "p4"  ;
NET "IOW0<1>"  LOC = "p5"  ;
NET "IOW0<2>"  LOC = "p8"  ;
NET "IOW0<3>"  LOC = "p9"  ;
NET "IOW0<4>"  LOC = "p11"  ;
NET "IOW0<5>"  LOC = "p12"  ;
NET "IOW0<6>"  LOC = "p15"  ;
NET "IOW0<7>"  LOC = "p16"  ;

NET "IOW1<0>"  LOC = "p18"  ;
NET "IOW1<1>"  LOC = "p19"  ;
NET "IOW1<2>"  LOC = "p22"  ;
NET "IOW1<3>"  LOC = "p23"  ;
NET "IOW1<4>"  LOC = "p24"  ;
NET "IOW1<5>"  LOC = "p25"  ;
NET "IOW1<6>"  LOC = "p28"  ;
NET "IOW1<7>"  LOC = "p29"  ;

NET "IOW2<0>"  LOC = "p31"  ;
NET "IOW2<1>"  LOC = "p33"  ;
NET "IOW2<2>"  LOC = "p34"  ;
NET "IOW2<3>"  LOC = "p35"  ;
NET "IOW2<4>"  LOC = "p36"  ;
NET "IOW2<5>"  LOC = "p39"  ;
NET "IOW2<6>"  LOC = "p40"  ;
NET "IOW2<7>"  LOC = "p41"  ;

NET "IOW3<0>"  LOC = "p42"  ;
NET "IOW3<1>"  LOC = "p45"  ;
NET "IOW3<2>"  LOC = "p47"  ;
NET "IOW3<3>"  LOC = "p48"  ;
NET "IOW3<4>"  LOC = "p49"  ;
```

```
NET "IOW3<5>"  LOC = "p50"  ;
NET "IOW3<6>"  LOC = "p55"  ;
NET "IOW3<7>"  LOC = "p56"  ;

NET "npreq"  LOC = "p192"  ; #B2
NET "nprd"  LOC = "p193"  ;  #B1
NET "npwr"  LOC = "p196"  ;  #B0
NET "IOAD<0>"  LOC = "p77"  ;#A7
NET "IOAD<1>"  LOC = "P185" ; #A6

NET "Ttest<0>"  LOC = "p3"  ;    #B4
NET "Ttest<1>"  LOC = "p200"  ;  #B5
NET "Ttest<2>"  LOC = "p199"  ;  #B6
NET "Ttest<3>"  LOC = "p197"  ;  #B7

NET "nmreq"  LOC = "p168"  ;
NET "nrd"  LOC = "p139"  ;
NET "nwr"  LOC = "p152"  ;
NET "nbhe"  LOC = "p138"  ;
NET "nble"  LOC = "p137"  ;

NET "IRtest<0>"  LOC = "p60"  ;
NET "IRtest<1>"  LOC = "p61"  ;
NET "IRtest<2>"  LOC = "p62"  ;
NET "IRtest<3>"  LOC = "p63"  ;
NET "IRtest<4>"  LOC = "p2"  ;
NET "IRtest<5>"  LOC = "p108"  ;
NET "IRtest<6>"  LOC = "p109"  ;
NET "IRtest<7>"  LOC = "p112"  ;
NET "IRtest<8>"  LOC = "p126"  ;
NET "IRtest<9>"  LOC = "p127"  ;
NET "IRtest<10>"  LOC = "p129"  ;
NET "IRtest<11>"  LOC = "p202"  ;
NET "IRtest<12>"  LOC = "p203"  ;
NET "IRtest<13>"  LOC = "p205"  ;
NET "IRtest<14>"  LOC = "p206"  ;
NET "IRtest<15>"  LOC = "p103"  ;
```

数码管 S5，S4：IRtest（当前指令）
数码管 S3-S0：IOW3-IOW0
拨码开关 K3-K0：IOR3-IOR0
LED 灯 B7-B4：Ttest（节拍）
LED 灯 B2：nPREQ
LED 灯 B1：nPRD
LED 灯 B0：nPWR
LED 灯 A7-A6：IOAD

六．处理器测试程序

指令地址	指令助记符	16 进制代码	说明
0000	MVI R0, 00000000	4000	R0 寄存器赋值 0
0001	OUT R0, S0	8000	S0 数码管显示 00
0002	MVI R1, 00000001	4101	R1 寄存器赋值 1
0003	OUT R1, S1	8101	S1 数码管显示 01
0004	MVI R2, 00000010	4202	R2 寄存器赋值 2
0005	OUT R2, S2	8202	S2 数码管显示 02
0006	MVI R7, 00000000	4700	R7 寄存器赋值 0
0007	OUT R7, S3	8703	S3 数码管显示 00
0008	STA R2, 01000000	6240	向地址 0040 写入数据 02
0009	LDA R3, 01000000	7340	取出地址 0040 数据至 R3
000A	OUT R3, S3	8303	S3 数码管显示 02
000B	JZ R0, 10000000	1080	跳转至地址 0080 处指令
0080	ADD R1, R2	3102	(R1)+(R2)→R1
0081	OUT R1, S1	8101	S1 数码管显示 03
0082	SUB R1, R2	2102	(R1)-(R2)→R1
0083	OUT R1, S1	8101	S1 数码管显示 01
0084	MOV R1, R2	5102	(R2)→R1
0085	OUT R1, S1	8101	S1 数码管显示 02
0086	IN R3, K3	9303	K3 输入数据 69 到寄存器 R3
0087	OUT R3, S3	8303	S3 数码管显示 69
0088	JMP 11110000	00F0	跳转至地址 00F0 处指令
00F0	STA R3, 01000001	6341	向地址 0041 写入数据 69
00F1	LDA R1, 01000001	7141	取出地址 0041 数据至 R1
00F2	OUT R1, S1	8101	S1 数码管显示 69
00F3	JMP 10000000	0080	跳转至地址 0080 处的指令

注：本指令序列在 FPGA 下载过程中使用，用于测试处理器的功能。通过 SD2100 型数字逻辑实验台测试，各指令的运行结果完全正确，处理器功能正常。

七．遇到的问题及解决方案

1.设计过程

问题 1：

在最初的设计中，指令译码由一个单独的译码模块完成并占用一个译码周期，降低处理器运行效率的问题。

解决方案：

指令译码使用简单的组合逻辑即可实现，因此可以将译码模块整合入运算管理模块中，在运算周期根据指令操作码的不同产生各种控制信号控制后续操作，这样便可以节省一个节拍。

问题 2：

在回写周期更新寄存器内容时判断寄存器的地址问题。

解决方案：

若使用向运算管理模块送寄存器地址的方式会增加管脚数量，实际上这是没有必要的，因为来自取指管理模块的指令输入 ir 在整个指令周期内是保持不变的，因此每次更新寄存器内容时无需送入待更新的寄存器地址，只需根据 ir 信号的第 10 到 8 位进行计算即可。在运算管理模块进程中使用 variable 类型表示寄存器地址。当进程被激活时，根据 ir 的内容计算寄存器的地址以便对其进行更新。

问题 3：

同一总线上不同情况下传输数据宽度不同的问题。

解决方案：

在设计过程中，有些总线上传输的数据在不同情况下具有不同的宽度。如存储管理模块与回写管理模块之间用于传递待回写内容的 write_i(write_o)总线，当需要回写到 PC 时其数据宽度为 16 位，而当需要回写到寄存器时其数据宽度为 8 位。同理，访存控制模块与主存储器之间的地址总线在取指阶段传输的是 16 位的指令，而在访存阶段传输的是 8 位数据。

为了解决这个问题，可以使用当数据宽度小于总线宽度时只使用总线的低位部分进行传输，而将高位部分置 0 的办法。

2.仿真过程

问题 1：

访存控制模块仿真过程中数据总线出现冲突变成'X'。

解决方案：

这是由于数据总线在非访存周期没有置高阻造成的，当数据总线仍为输出状态时，若试图向其输入数据便会出现冲突，解决方法是在仿真过程中数据总线仅在取指周期和访存周期进行输入输入，其余时间置为高阻态。

3.下载过程

问题 1：

在实验台上进行操作时，PC 无法完成自动加 1，反复取出第一条指令。

解决方案：

初始设计中 PC 在取指周期的节拍下降沿完成加 1，虽然仿真能成功但是在实验台上无法完成。解决方案是将时钟控制模块的节拍信号 t 连接到取指管理模块，当 t(1)上升沿到来时（即运算周期开始时）将 PC 加 1。

```
process(rst, t(1), pc_update, pc_i)
    begin
        if (rst = '1') then
            pc <= (others=>'0');
        elsif pc_update = '1' then
            pc <= pc_i;
        elsif rising_edge(t(1)) then
            pc <= pc + 1;
        end if;
    end process;
```

问题 2：

指令取出后仅在取指周期内保持，导致后续操作无法进行。

解决方案：

在实验台上进行开发时，数码管 S5 与 S4 显示的 IR 内容仅在取值周期内可见，再次点击单拍键后便会消失，后续操作也会失败，由此可见指令被取出后没有在指令周期内保持。解决方案是将系统时钟信号 clk 连接到取指管理模块，在节拍信号 t 仍为“0001”时遇到的第一个时钟下降沿处将 IR 内容输出，之后保持。这样可以保证 IR 输出的是本次取指周期取出的指令。

```
process(clk, t, ir)
    begin
        if (falling_edge(clk) and t = "0001") then
            ir_o <= ir;
        end if;
    end process;
```

问题 3：

回写寄存器或 PC 出错，当需要回写到某个寄存器或者 PC 时写入的内容不是想要的结果。

解决方案：

这是由于回写管理模块的更新信号与更新内容不同步造成的，当 reg_update 信号与 pc_update 信号有效时，运算管理模块或取指管理模块会迅速对寄存器或 PC 进行更新，若此时传输更新内容的总线上的数据仍未更新，便会写入错误的内容。

解决方案是将更新信号与更新内容信号的赋值分离，将更新内容 pc_new 和 reg_new 通过组合逻辑直接与 write_i 连接在一起，而更新信号 pc_update 与 reg_update 则通过进程来进行赋值，仅在回写周期内有效。

八．心得与体会

①硬件程序的设计，应该严格遵循相应的开发步骤，使用自顶向下的思想，从系统整体要求出发，将设计内容逐渐细化，最后完成需求。自顶向下的方法可以保持逻辑清晰。
②FPGA 芯片功能灵活可扩展，未来将成为硬件发展的趋势。要掌握集成的 PLD/FPGA 开发环境（Xilinx 等），了解 FPGA 开发的基本流程：系统划分->代码输入->编译->代码功能仿真->综合->芯片仿真。
③更具体的掌握了 VHDL 语言。其模块内部的信号，应该尽可能精简，输入输出信号减少对 inout 类型的使用，掌握如何书写并行、顺序语句、条件控制语句。
④对 VHDL 本质上是并发的这一特性有了更加深刻的理解，体会到了 VHDL 与高级编程语言的区别和联系。
⑤下载调试过程中需要综合考虑实验台的实际特性对代码进行调整。仿真成功并不意味着下载成功，调试过程中一定要保持耐心。
⑥通过本课程的学习我熟悉了 FPGA 芯片的相关操作，对以后硬件系统设计打下了基础。