哈爾濱工業大學

课程报告

课程名称：计算机组成原理

报告题目：设计 RISC 处理器

所在院系：软件学院

所在专业：软件工程

学生姓名：石卓凡

学生学号：120L021011

选课时间：2022 年秋季学期

评阅成绩：

目录

加分项 1：IO 接口，加分项 2：verilog 设计并仿真 均已完成

一、指令格式设计；

指令类型	指令	指令说明	指令格式
非访存指令	ADD Ri, Rj	加法指令 Ri + Rj → Ri	00000 XXX(Ri) XXX(Rj) 00000
	SUB Ri, Rj	减法指令 Ri -Rj → Ri	00001 XXX(Ri) XXX(Rj) 00000
	MOV Ri, Rj	寄存器传送指令 Rj → Ri	00010 XXX(Ri) XXX(Rj) 00000
	MVI Ri, X	立即数传送指令 X → Ri	00011 XXX(Ri) YYYYYYYY(立即数 X)
访存指令	STA Ri, X	存数指令 Ri → [R7//X]	00100 XXX(Ri) YYYYYYYY(立即数 X)
	LDA Ri, X	取数指令 [R7//X] → Ri	00101 XXX(Ri) YYYYYYYY(立即数 X)
转移类指令	JZ Ri, X	条件转移（零则转）指令 if (Ri = 0) then [R7//X] → PC	00110 XXX(Ri) YYYYYYYY(立即数 X)
	JMP X	无条件转移指令 [R7//X] → PC	00111 000 YYYYYYYY(立即数 X)

I/O 指令	IN Ri, PORT	输入指令 [PORT] → Ri	01000 XXX(Ri) ZZZZZZZZ(PORT)
	OUT Ri, PORT	输出指令 Ri → [PORT]	01001 XXX(Ri) ZZZZZZZZ(PORT)

二、微操作的定义；

（一）取指阶段

M(PC)->IR	从主存储器中根据 PC 地址取出指令放入 IR
1->R	发出读命令
1 ->Mem	向主存发出命令
PC+1->PC	先假定不是跳转指令，形成下一跳指令地址 若是跳转指令，会在写回阶段重新用 PC_jump_data 重新更新下一跳指令地址

（二）执行阶段

ADD Ri, Rj 加法指令 Ri + Rj → Ri	Reg(Ad1(IR)) + Reg(Ad2(IR))->Reg(Ad1(IR))
SUB Ri, Rj 减法指令 Ri -Rj → Ri	Reg(Ad1(IR)) - Reg(Ad2(IR))->Reg(Ad1(IR))
MOV Ri, Rj 寄存器传送指令 Rj → Ri	Reg(Ad2(IR))->Reg(Ad1(IR))
MVI Ri, X 立即数传送指令 X → Ri	X->Reg(Ad1(IR))
STA Ri, X 存数指令 Ri → [R7//X]	Reg(Ad1(R1)) -> M(Reg(R7)//Ad2(IR)) 1 -> W 1 -> Mem
LDA Ri, X 取数指令 [R7//X] → Ri	M(Reg(R7)//Ad2(IR)) -> Reg(Ad1(IR)) 1 - > R 1 -> Mem
JZ Ri, X 条件转移（零则转）指令 if (Ri = 0) then [R7//X]	[Zero(Reg(Ad1(IR))) * Reg(R7)//Ad2(IR) + Nzero(Reg(Ad1(IR))] +PC -> PC

→ PC	
JMP X 无条件转移指令 [R7//X] → PC	Reg(R7)//Ad2(IR) + PC -> PC
IN Ri, PORT 输入指令 [PORT] → Ri	M(Ad(IR)) -> Reg(Ad1(IR)) 1 -> R 1 ->IO
OUT Ri, PORT 输出指令 Ri → [PORT]	Reg(Ad1(IR))-> M(Ad(IR)) 1 -> R 1 ->IO

说明：

- 对于 CPU 外界,包括了存储器和 IO 接口
- 对读写的 IO 端口的操作相当于一个“存储器”,然后 M/IO 去选择读写哪个“存储器”
- Ad1(IR)取得是 IR[10:8],对应的是 Ri 位置
- Ad2(IR)取得是 IR[7:5]，对应的是 Rj 位置
- Ad(IR)取得是 IR[7:0]，对应的是立即数 X 位置
- Reg(R7)依靠操作码判断，指定在 JMP 指令下的 Rj 对应的是 R7，指定在 JMP 指令下的 IR[7:5]对应的是 111
- 1 -> W,指的是向 CPU 外界发出的是写信号 W,对应的 R/W 信号为写
- 1 -> R,指的是向 CPU 外界发出的是读信号 R,对应的 R/W 信号为读
- 1 -> Mem,指的是向 CPU 外界读写的是存储器 Memory,对应的 M/IO 信号为存储器 M
- 1 ->IO,指的是向 CPU 外界读写的是 IO 接口,对应的 M/IO 信号为 IO 接口
- Zero(Reg(Ad1(IR)))代表，Ri 如果是 0，Zero(Reg(Ad1(IR)))则为 1
- Nzero(Reg(Ad1(IR)))代表，Ri 如果不是 0，Nzero(Reg(Ad1(IR)))则为 1

三、节拍的划分；

指令	**取指阶段**		**执行阶段**			
指令	取指周期 T0	取指周期 T1	译码周期 T0	执行周期 T1	访存周期 T2	写回周期 T3
ADD	M(PC)->	PC+1	Reg(Ad1(IR))	valA+val	空白	ALUOUT ->

Ri, Rj 加法指令 Ri + Rj → Ri	IR 1->R 1 ->Mem	->PC	-> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	B -> ALUOUT		Reg(Ad1(IR))
SUB Ri, Rj 减法指令 Ri -Rj → Ri	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	valA-valB -> ALUOUT	空白	ALUOUT -> Reg(Ad1(IR))
MOV Ri, Rj 寄存器传送指令 Rj → Ri	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	valB -> ALUOUT	空白	ALUOUT -> Reg(Ad1(IR))
MVI Ri, X 立即数传送指令 X → Ri	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	X -> ALUOUT	空白	ALUOUT -> Reg(Ad1(IR))
STA Ri, X 存数指令 Ri → [R7//X]	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	valA -> ALUOUT	ALUOUT->M(Addr) 1->W 1 -> Mem	空白
LDA Ri, X 取数指令 [R7//X]	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR	空白	M(Addr)->Rtemp 1->R 1 -> Mem	Rtemp -> Reg(Ad1(IR))

→ Ri			) -> Addr Ad(IR) -> X			
JZ Ri, X 条件转移（零则转）指令 if (Ri = 0) then [R7//X] → PC	M(PC)-> IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	ValA -> ALUOUT	空白	If ALUOUT==0 Then Addr->PC
JMP X 无条件转移指令 [R7//X] → PC	M(PC)-> IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	空白	空白	Addr->PC
IN Ri, PORT 输入指令 [PORT] → Ri	M(PC)-> IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	空白	M(X)->Rtemp 1 -> R 1 -> IO	Rtemp -> Reg(Ad1(IR))
OUT Ri, PORT 输出指令 Ri → [PORT]	M(PC)-> IR 1->R 1 ->Mem	PC+1 ->PC	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X	valA -> ALUOUT	ALUOUT ->M(X) 1->W 1 -> IO	空白

（一）取指阶段

1．取指周期 T0

指令	取指周期 T0	取指周期 T1
所有指令	M(PC)->IR 1->R 1 ->Mem	PC+1 ->PC

2．取指周期 T1

指令	取指周期 T1
所有指令	PC+1 ->PC

3．取指周期 T2

指令	取指周期 T2
所有指令	空白

4．取指周期 T3

指令	取指周期 T3
所有指令	空白

（二）执行阶段

1．译码周期 T0

指令	**译码周期 T1**
所有指令	Reg(Ad1(IR)) -> valA Reg(Ad2(IR)) -> valB Reg(R7)//Ad(IR) -> Addr Ad(IR) -> X

2．运算周期 T1

指令	**运算周期 T1**
ADD　Ri, Rj 加法指令 Ri + Rj → Ri	valA+valB -> ALUOUT
SUB　Ri, Rj 减法指令 Ri -Rj → Ri	valA-valB -> ALUOUT
MOV　Ri, Rj 寄存器传送指令 Rj → Ri	valB -> ALUOUT
MVI　Ri, X 立即数传送指令 X → Ri	Ad(IR) -> ALUOUT
STA　Ri, X 存数指令 Ri → [R7//X]	valA -> ALUOUT
LDA　Ri, X 取数指令 [R7//X] → Ri	空白
JZ　Ri, X 条件转移（零则转）指令 if (Ri = 0) then [R7//X] → PC	ValA -> ALUOUT
JMP　X 无条件转移指令 [R7//X] → PC	空白
IN　Ri, PORT 输入指令 [PORT] → Ri	空白
OUT　Ri, PORT 输出指令 Ri → [PORT]	valA -> ALUOUT

3．访存周期 T2

指令	**访存周期 T2**
ADD　Ri, Rj	空白

加法指令 Ri + Rj → Ri	
SUB Ri, Rj 减法指令 Ri -Rj → Ri	空白
MOV Ri, Rj 寄存器传送指令 Rj → Ri	空白
MVI Ri, X 立即数传送指令 X → Ri	空白
STA Ri, X 存数指令 Ri → [R7//X]	ALUOUT->M(Addr) 1->W 1 -> Mem
LDA Ri, X 取数指令 [R7//X] → Ri	M(Addr)->Rtemp 1->R 1 -> Mem
JZ Ri, X 条件转移（零则转）指令 if (Ri = 0) then [R7//X] → PC	空白
JMP X 无条件转移指令 [R7//X] → PC	空白
IN Ri, PORT 输入指令 [PORT] → Ri	M(X)->Rtemp 1 -> R 1 -> IO
OUT Ri, PORT 输出指令 Ri → [PORT]	ALUOUT ->M(X) 1 ->W 1 -> IO

4．写回周期 T3

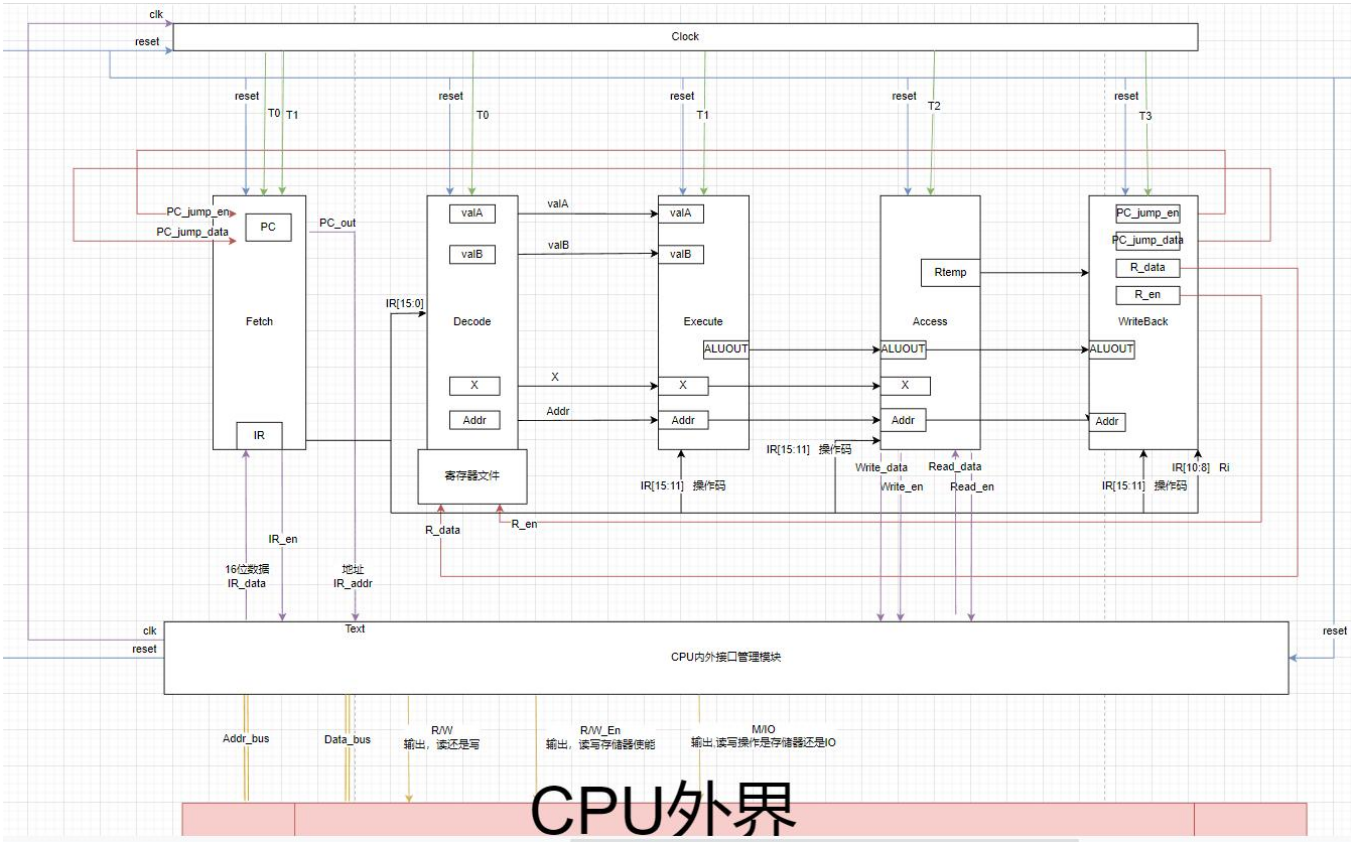
指令	**写回周期 T3**
ADD Ri, Rj 加法指令 Ri + Rj → Ri	ALUOUT -> Reg(Ad1(IR))
SUB Ri, Rj	ALUOUT ->

减法指令 Ri -Rj → Ri	Reg(Ad1(IR))
MOV Ri, Rj 寄存器传送指令 Rj → Ri	ALUOUT -> Reg(Ad1(IR))
MVI Ri, X 立即数传送指令 X → Ri	ALUOUT -> Reg(Ad1(IR))
STA Ri, X 存数指令 Ri → [R7//X]	空白
LDA Ri, X 取数指令 [R7//X] → Ri	Rtemp -> Reg(Ad1(IR))
JZ Ri, X 条件转移（零则转）指令 if (Ri = 0) then [R7//X] → PC	If ALUOUT==0 Then Addr->PC
JMP X 无条件转移指令 [R7//X] → PC	Addr->PC
IN Ri, PORT 输入指令 [PORT] → Ri	Rtemp -> Reg(Ad1(IR))
OUT Ri, PORT 输出指令 Ri → [PORT]	空白

四、处理器结构设计框图及功能描述；

（一）处理器结构设计框图

原图具体打印出来了附在后面



（二）处理器模块划分

处理器模块划分	Clock 模块
	CPU 内外接口管理模块
	Fetch 模块
	Decode 模块
	Execute 模块
	Access 模块
	WriteBack 模块

（三）各模块说明及功能描述

1．Clock 模块

功能说明：根据 clk 输入以此产生有效的时钟周期 T0,T1,T2,T3

信号说明：

输入或输出	信号	位数	来源	去向	说明

In	Clk	1	CPU 内外接口管理模块		时钟信号
	Reset	1	CPU 内外接口管理模块		复位重置信号
Out	T0	1		Fetch 取值模块，Decode 译码模块	时钟节拍
	T1	1		Decode 译码模块	时钟节拍
	T2	1		Execute 运算模块	时钟节拍
	T3	1		Access 访问模块	时钟节拍

2．CPU 内外接口管理模块

功能说明：处理 CPU 内部与外界的连接接口管理，外界包括了主存，IO 端口。
对于 CPU 外界,包括了存储器和 IO 接口
对读写的 IO 端口的操作相当于一个“存储器”,然后 M/IO 去选择读写哪个“存储器”
1.负责 CPU 对外界的读写交互的管理，在 clk 上升沿时期可以根据 RW，en，MemIO 来对主存储器或者 IO 接口进行读写
2.模拟着主存储器和 IO 接口数据的内容,模拟了十条主存储器地址从 0000_0000_0000_0000 到 0000_0000_0000_1001 的内容,模拟了 IO 接口数据地址为 0000_0000_1111_1111

信号说明：

输入或输出	信号	位数	来源	去向	说明
总线	Data_bus	8/16		CPU 外界	数据总线， 当从主存中提取 16 位的指令信息，直接 16 位 当与主存进行 8 位的数据交换，将高 8 位设置为 00000000，使得数据变为 16 位
	Addr_bus	16		CPU 外界	地址总线
In	IR_addr	16	Fetch 模块		PC 想要读取的当前指令的地址（存储着之后放入 IR 中的数

					据）
	IR_en	1	Fetch 模块		Fetch 模块发出的是否读取 PC 下一指令地址的指令
	Write_data	8	Access 模块		模块想要向外界写的数据
	Write_en	1	Access 模块		模块发出的与外界的写信号，将改变输出的 R/W 信号
	Read_en	1	Access 模块		模块发出的与外界的读信号，将改变输出的 R/W 信号
Out	M/IO	1		CPU 外界	M/IO 输出,读写操作是存储器还是 IO
	R/W_En	1		CPU 外界	R/W_En 输出至 CPU 外界，读写存储器使能
	R/W	1		CPU 外界	R/W 输出至 CPU 外界，读还是写
	Clk	1		其他所有模块	时钟信号
	Reset	1		其他所有模块	复位重置信号
	Read_data	8		Access 模块	模块向 CPU 外界读入的数据

3．Fetch 模块

功能说明：负责取指
T0 周期：RW 设为 0 代表 R，使能端 en 设为 1 有效，MemIO 设为 0 代表 Mem，向外输出 PC_out,通过 PC_out 来向 CPU 接口（调用 CPU_export 模块）给出指令地址，由 CPU 接口与主存储器进行读写，然后返回 IR 给 CPU 接口，再返回 IR 给 fetch 模块
T1 周期：将 IR 输出给其他模块
执行阶段（Decode 模块）
模块中保存了寄存器文件，管理着 R0~R7

信号说明：

输 入	信号	位数	来源	去向	说明

或输出					
In	Reset	1	CPU 内外接口管理模块		复位重置信号
	T0	1	Clock 模块		时钟节拍
	T1	1	Clock 模块		时钟节拍
	PC_jump_en	1	WriteBack 模块		跳转指令更新 PC 的使能端
	PC_jump_data	1	WriteBack 模块		跳转指令更新 PC 的跳转地址数据
	IR_data	16	CPU 内外接口管理模块		16 位指令数据
Out	PC_out	16		CPU 内外接口管理模块	PC 输出的 16 位指令地址
	IR_en	1		CPU 内外接口管理模块	读取指令使能端
	IR	16		Decode,Execute,Access, WriteBack 模块	16 位指令数据

4．Decode 模块

功能说明：负责译码

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

信号说明：

输入或输出	信号	位数	来源	去向	说明
in	Reset	1	CPU 内外接口管理模块		复位重置信号
	T0	1	Clock 模块		时钟节拍
	IR[15:0]	16	Fetch 模块		16 位指令数据
	R_data	8	WriteBack 模块		传入寄存器文件的写数据
	R_en	1	WriteBack 模块		传入寄存器文件的写数据的使能
Out	valA	8		Execute 模块	根据 Reg(IR[10:8])读取到的第一个寄存器 Ri 的值
	valB	8		Execute 模	根据 Reg(IR[7:5])读取到的第二个寄存器 Rj

				块	的值
	X	8		Execute 模块	根据 IR[7:0]读取到的立即数 X
	Addr	16		Execute 模块	8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩充寻址生成 Addr 地址

5. Execute 模块

功能说明：负责运算，执行
T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

信号说明：

输入或输出	信号	位数	来源	去向	说明
In	Reset	1	CPU 内外接口管理模块		复位重置信号
	T1	1	Clock 模块		时钟节拍
	IR[15:11]	5	Fetch 模块		5 位的指令操作码
	valA	8	Execute 模块		根据 Reg(IR[10:8]) 读取到的第一个寄存器 Ri 的值
	valB	8	Execute 模块		根据 Reg(IR[7:5]) 读取到的第二个寄存器 Rj 的值
	X	8	Execute 模块		根据 IR[7:0] 读取到的立即数 X
	Addr	16	Execute 模块		8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩充寻址生成 Addr 地址
Out	ALUOUT	8		Access 模块	经过 ALU 相应的运算得来的结果
	Addr	16		Access 模块	8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩

					充寻址生成Addr地址

6．Access 模块

功能说明：负责访存

T2 周期：根据输入的 IR，valA，valB,X，Addr，根据不同的指令，如果是需要读写主存储器或者 IO 接口的，会通过 CPU 接口（调用 CPU_export 模块）设置 RW 读写位，en 使能端，MemIO 主存储器或接口端，为相应的数值，如果是读则读取的值会在 data_read,如果是写则将写入的值在 data_write。

信号说明：

输入或输出	信号	位数	来源	去向	说明
In	Reset	1	CPU 内外接口管理模块		复位重置信号
	T2	1	Clock 模块		时钟节拍
	IR[15:11]	5	Fetch 模块		5 位的指令操作码
	ALUOUT	8	Execute 模块		经过 ALU 相应的运算得来的结果
	Addr	16	Execute 模块		8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩充寻址生成 Addr 地址
	X	8	Execute 模块		根据 IR[7:0]读取到的立即数 X
Out	Write_data	8			向存储器写入的数据
	Write_en	1			向存储器发出写信号的使能端 1->W
	Read_data	8			从存储器读到的数据
	Read_en	1			向存储器发出读信号的使能端 1->R

	ALUOUT	8			经过 ALU 相应的运算得来的结果
	Addr	16			8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩充寻址生成 Addr 地址
	Rtemp	8			临时寄存器，用来暂存数据

7．WriteBack 模块

功能说明：负责写回

T3 周期：根据输入的 IR,ALUOUT,Rtemp,Addr,T3，根据不同的指令将需要更新的数值更新对应的寄存器，则会输出 R_select 写入的寄存器的编号,R_data 写入的寄存器的值,R_en 写入的使能端

如果是需要 JUMP 之类的跳转指令修改 PC，则会输出 PC_jump_en 跳转

信号说明：

输入或输出	信号	位数	来源	去向	说明
In	T3	1	CPU 内外接口管理模块		时钟信号
	Reset	1	CPU 内外接口管理模块		复位重置信号
	IR[15:11]	5	Fetch 模块		5 位的指令操作码
	IR[10:8]	3	Fetch 模块		指定 Ri
	ALUOUT	8	Access 模块		经过 ALU 相应的运算得来的结果
	Addr	16	Access 模块		8 位形式地址 X 经 R7 充当扩充寻址寄存器，扩充寻址生成 Addr 地址
Out	PC_jump_data	16		Fetch 模块	跳转指令要跳转的地址数据
	PC_jump_en	1		Fetch 模块	是否跳转指令更新 PC 的使能

	R_data	8		Fetch 模块	寄存器文件的更新寄存器的数据
	R_en	1		Fetch 模块	寄存器文件的更新寄存器的使能

五、如采用组合逻辑设计，列出操作时间表，画出每个控制信号的逻辑图；

（一）操作时间表

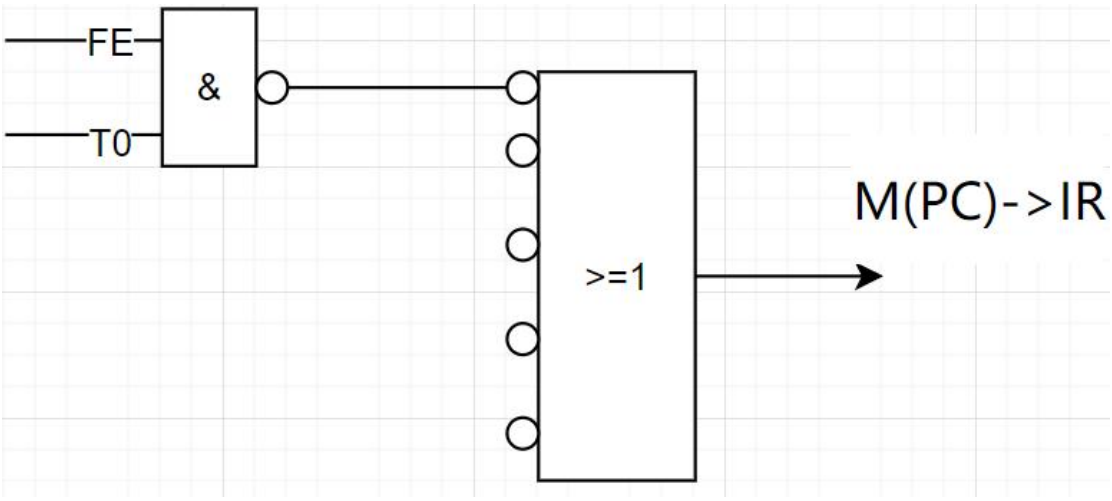
工作周期标记	划分	节拍	微操作命令信号	ADD Ri, Rj	SUB Ri, Rj	MOV Ri, Rj	MVI Ri, X	STA Ri, X	LDA Ri, X	JZ Ri, X	JMP X	IN Ri, PORT	OUT Ri, PORT
取指阶段 FE	Fetch 取指	T0	M(PC)->IR	1	1	1	1	1	1	1	1	1	1
		T0	1->R	1	1	1	1	1	1	1	1	1	1
		T0	1 ->Mem	1	1	1	1	1	1	1	1	1	1
		T1	PC+1 ->PC	1	1	1	1	1	1	1	1	1	1
执行阶段 EX	Decode 译码	T0	Reg(Ad1(IR)) -> valA	1	1	1	1	1	1	1	1	1	1
		T0	Reg(Ad2(IR)) -> valB	1	1	1	1	1	1	1	1	1	1
		T0	Ad(IR) -> X	1	1	1	1	1	1	1	1	1	1
		T0	Reg(R7)//Ad(IR) -> Addr	1	1	1	1	1	1	1	1	1	1
	Execute 执行	T1	valA+valB -> ALUOUT	1									
		T	valA-valB ->		1								

		1	ALUOUT										
		T1	valB -> ALUOUT			1							
		T1	X -> ALUOUT				1						
		T1	valA -> ALUOUT					1		1			1
		T1	空白						1		1		
	Access 访存	T2	ALUOUT->M(Addr)					1					
		T2	1->W					1					1
		T2	1 -> Mem					1	1				
		T2	M(Addr)->Rtemp						1				
		T2	1->R						1			1	
		T2	空白	1	1	1	1			1	1		
		T2	M(X)->Rtemp									1	
		T2	1 -> IO									1	1
		T2	ALUOUT ->M(X)										1
	Write Back 写回	T3	ALUOUT -> Reg(Ad1(IR))	1	1	1	1						
		T3	空白					1					1
		T3	Rtemp -> Reg(Ad1(IR))						1			1	
		T3	If ALUOUT==0 Then Addr->PC							1			
		T3	Addr->PC								1		

（二）每个控制信号逻辑图

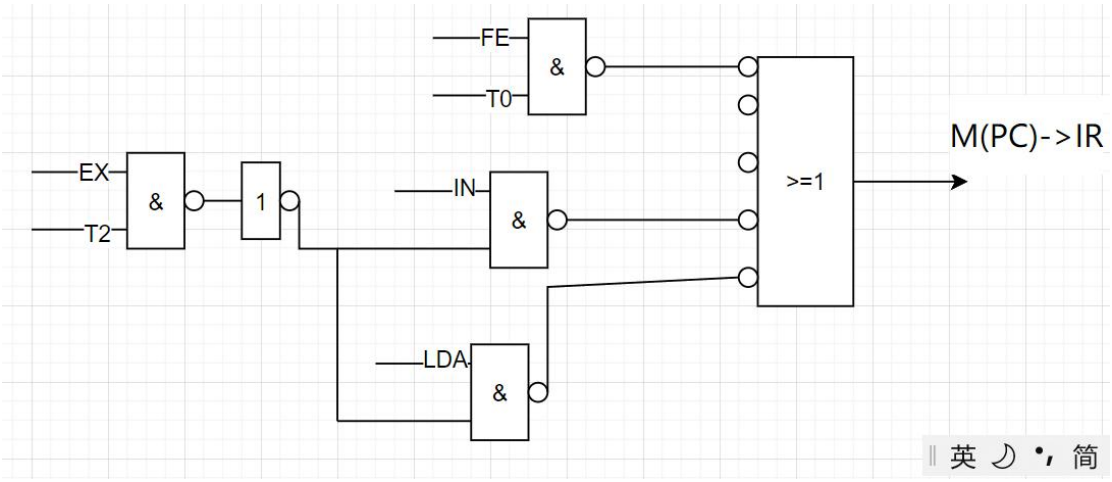
1．FE----取指周期

控制信号:
M(PC)->IR
微操作命令的最简表达式:
=FE*T0
控制信号逻辑图:



控制信号:
1 ->R
微操作命令的最简表达式:
=FE*T0 + EX*T2(LDA+IN)
控制信号逻辑图:

控制信号:
1 ->W
微操作命令的最简表达式:
=EX*T2(OUT+STA)
控制信号逻辑图:



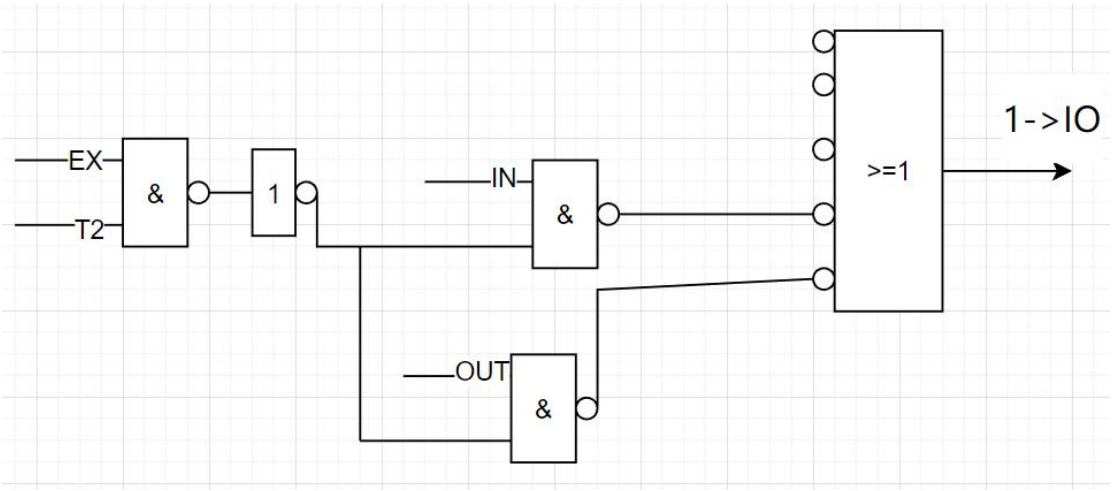
控制信号:

1 ->IO

微操作命令的最简表达式:

=EX*T2(IN+OUT)

控制信号逻辑图:



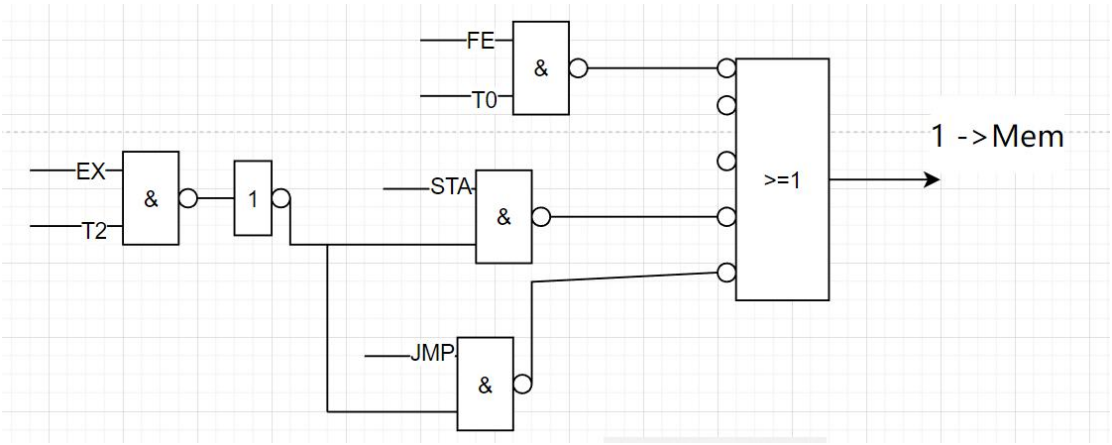
控制信号:

1 ->Mem

微操作命令的最简表达式:

=FE*T0 + EX*T2(STA+JMP)

控制信号逻辑图:



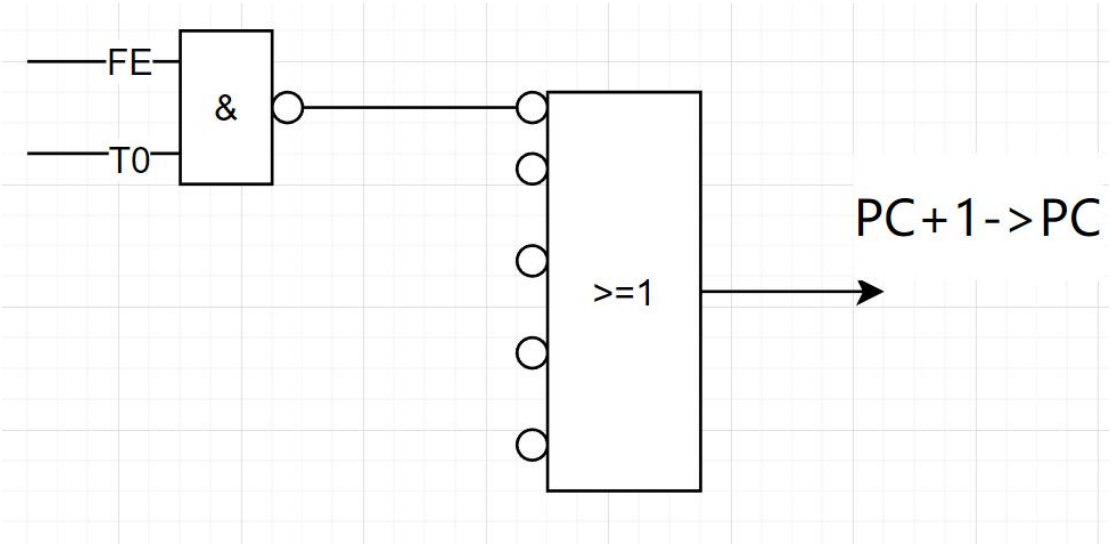
控制信号:

PC+1 ->PC

微操作命令的最简表达式:

=FE*T0

控制信号逻辑图:



2. EX----译码周期

控制信号:

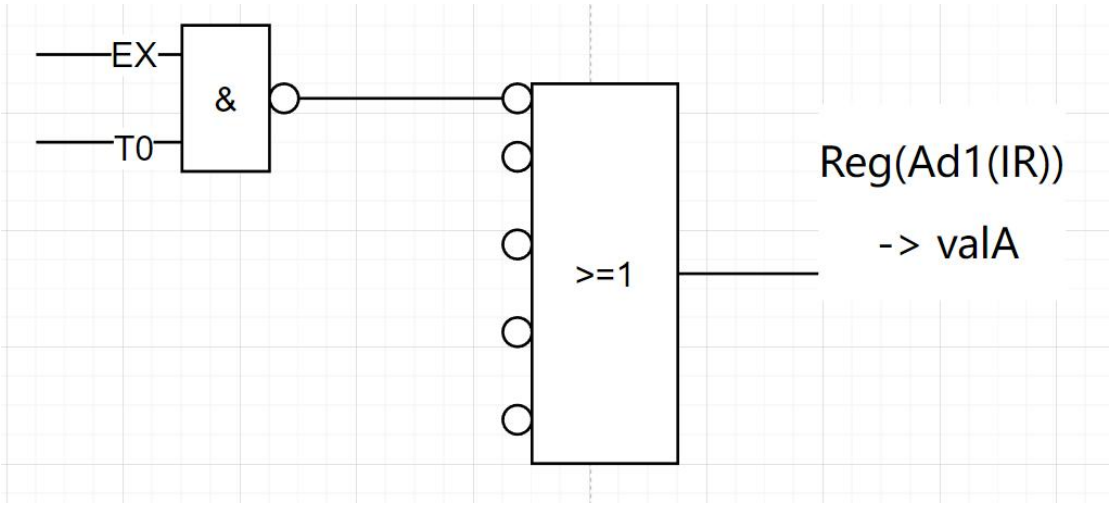
Reg(Ad1(IR))

-> valA

微操作命令的最简表达式:

=EX*T0

控制信号逻辑图:



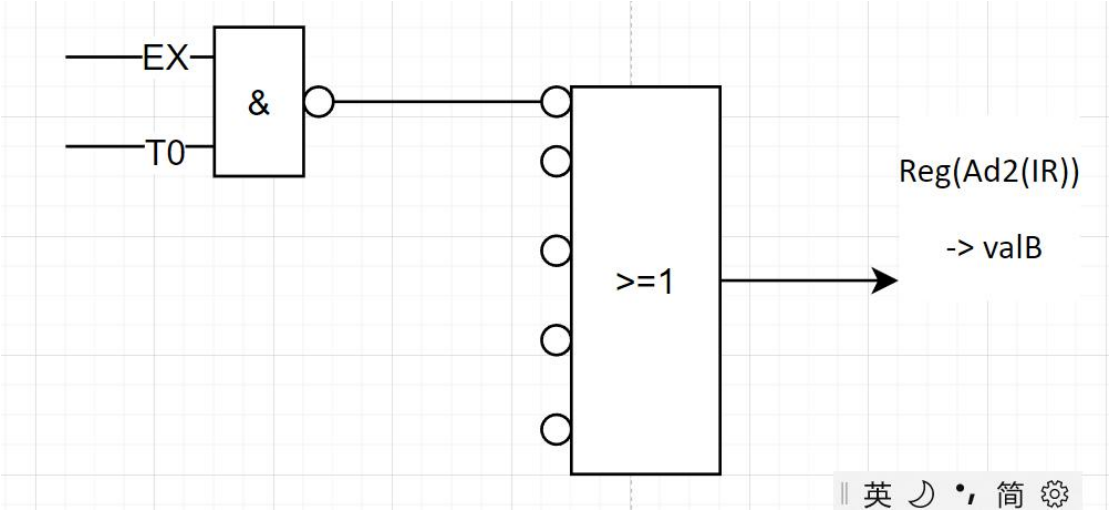
控制信号:

Reg(Ad2(IR))
  -> valB

微操作命令的最简表达式:

=EX*T0

控制信号逻辑图:



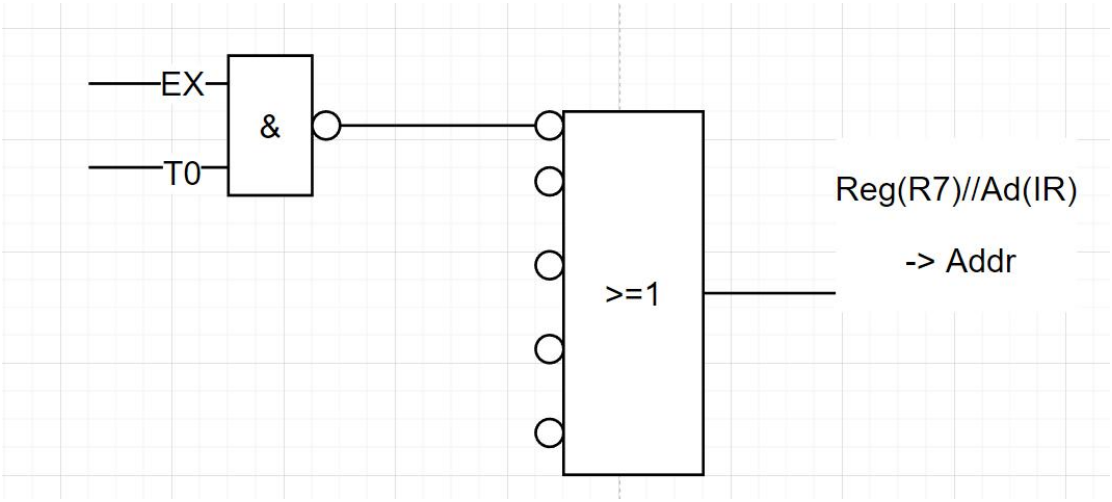
控制信号:

Reg(R7)//Ad(IR)
  -> Addr

微操作命令的最简表达式:

=EX*T0

控制信号逻辑图:



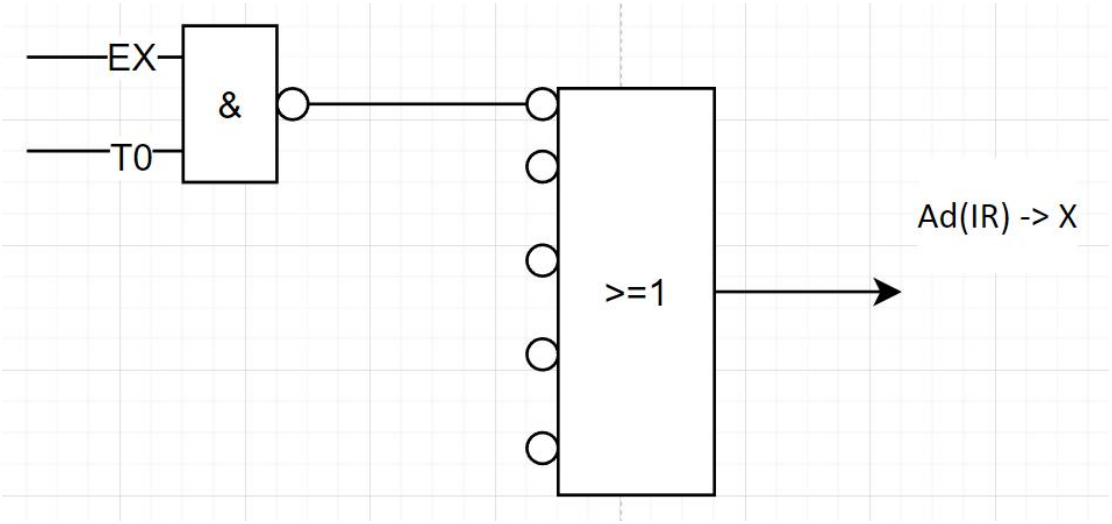
控制信号:

Ad(IR) -> X

微操作命令的最简表达式:

=EX*T0

控制信号逻辑图:



3. EX----执行周期

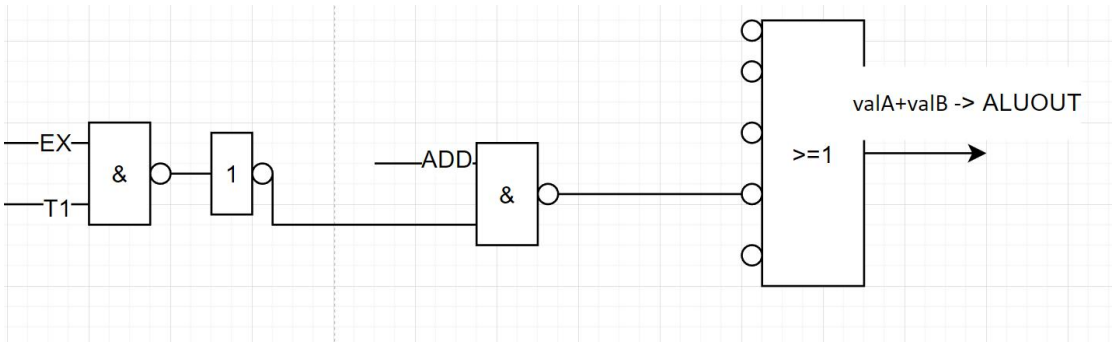
控制信号:

valA+valB -> ALUOUT

微操作命令的最简表达式:

=EX*T1*(ADD)

控制信号逻辑图:



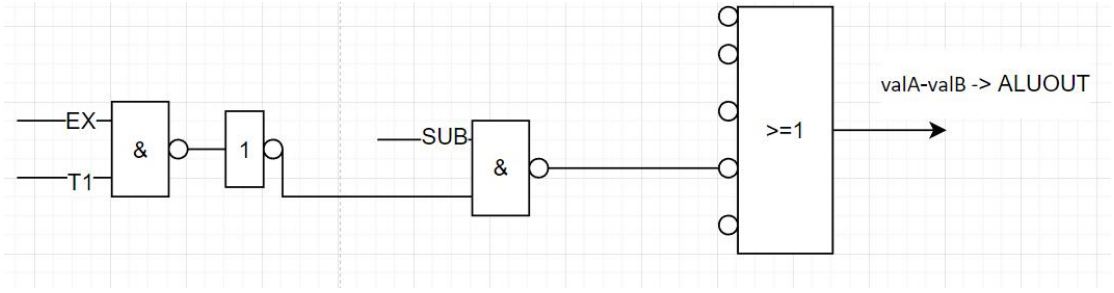
控制信号:

valA-valB -> ALUOUT

微操作命令的最简表达式:

=EX*T1*(SUB)

控制信号逻辑图:



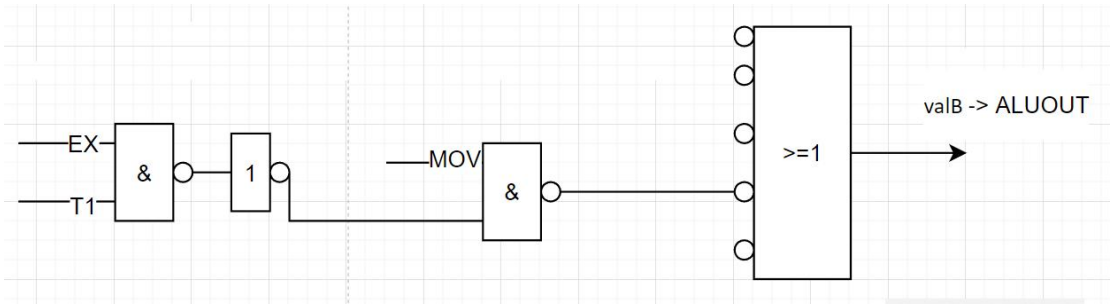
控制信号:

valB -> ALUOUT

微操作命令的最简表达式:

=EX*T1*(MOV)

控制信号逻辑图:



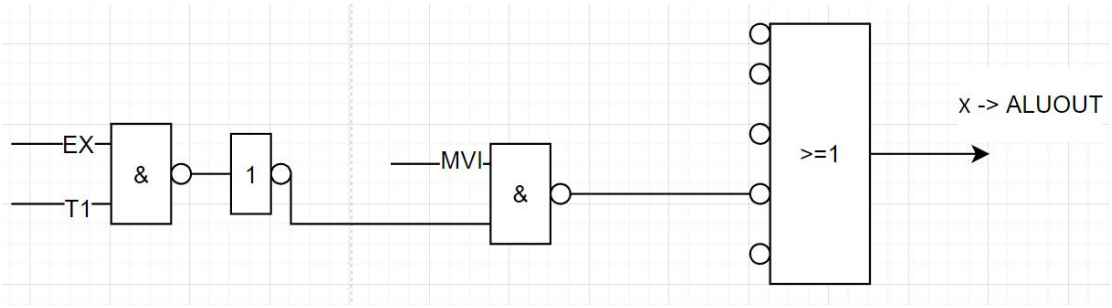
控制信号:

X -> ALUOUT

微操作命令的最简表达式:

=EX*T1*(MVI)

控制信号逻辑图:



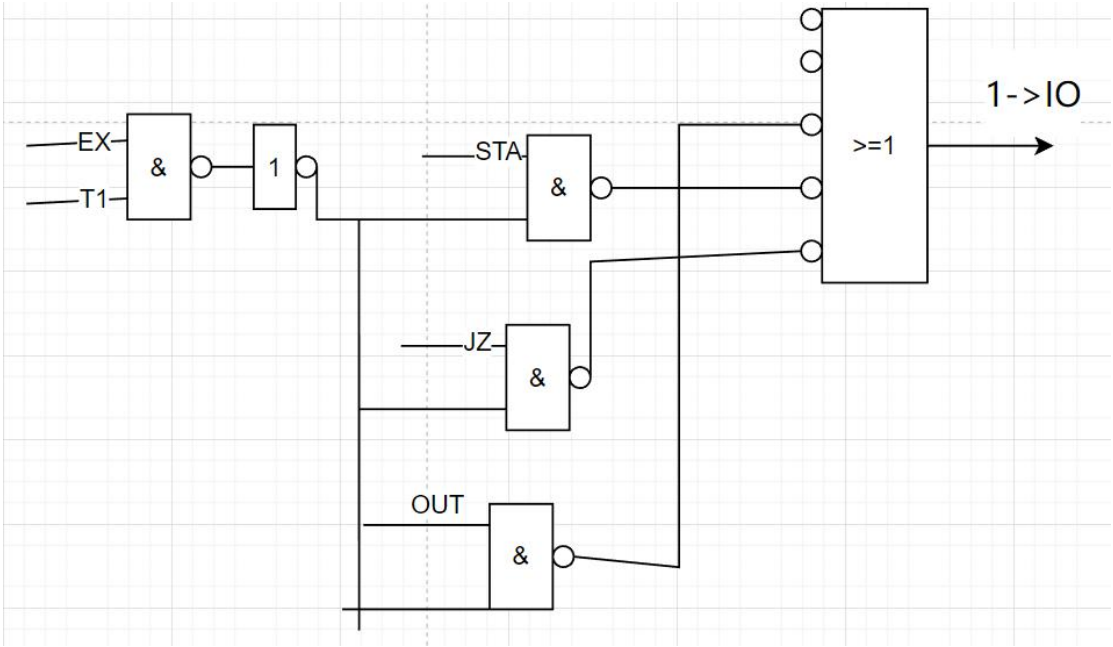
控制信号:

valA -> ALUOUT

微操作命令的最简表达式:

=EX*T1*(STA+JZ+OUT)

控制信号逻辑图:



4. EX----访存周期

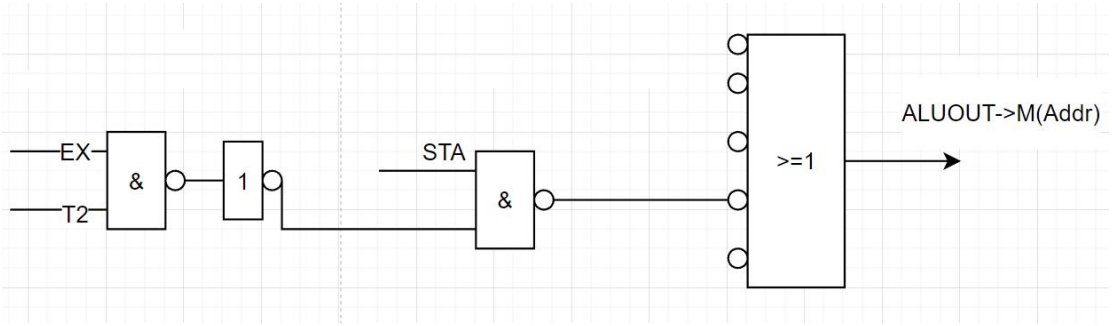
控制信号:

ALUOUT->M(Addr)

微操作命令的最简表达式:

=EX*T2*（STA)

控制信号逻辑图:



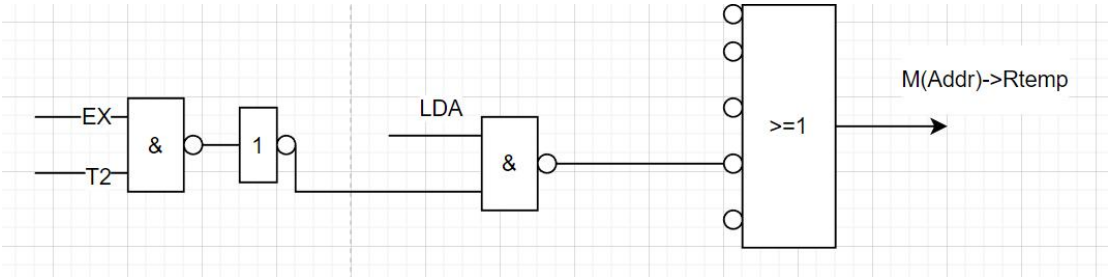
控制信号:

M(Addr)->Rtemp

微操作命令的最简表达式:

=EX*T2*（LDA)

控制信号逻辑图:



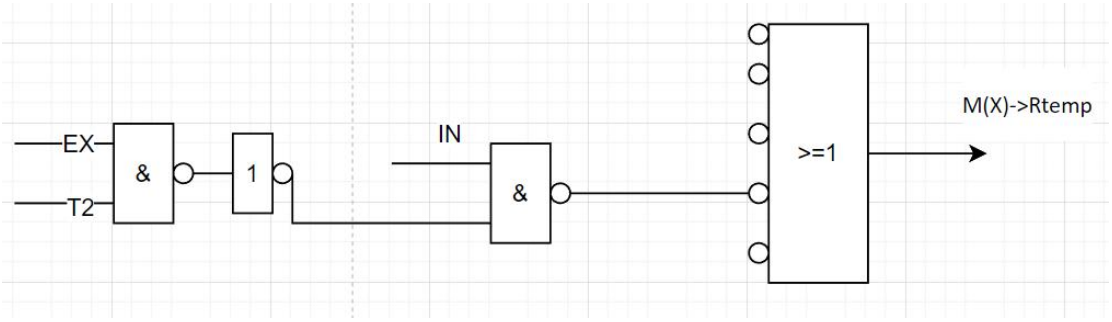
控制信号:

M(X)->Rtemp

微操作命令的最简表达式:

=EX*T2*（IN)

控制信号逻辑图:



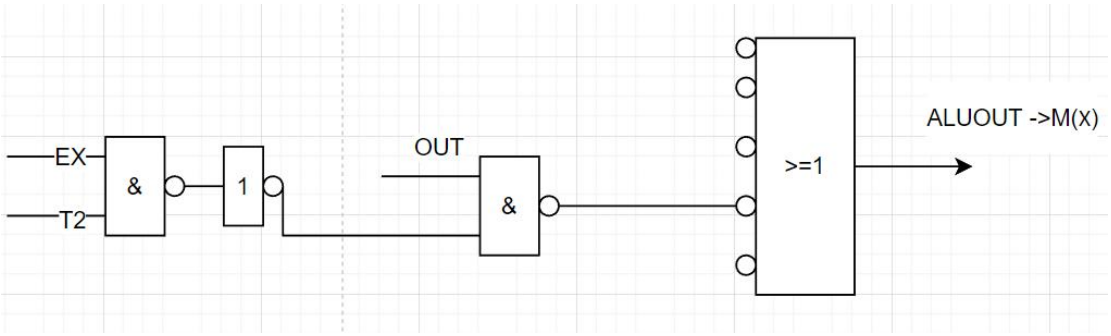
控制信号:

ALUOUT ->M(X)

微操作命令的最简表达式:

=EX*T2*（OUT)

控制信号逻辑图:



5．EX----写回阶段

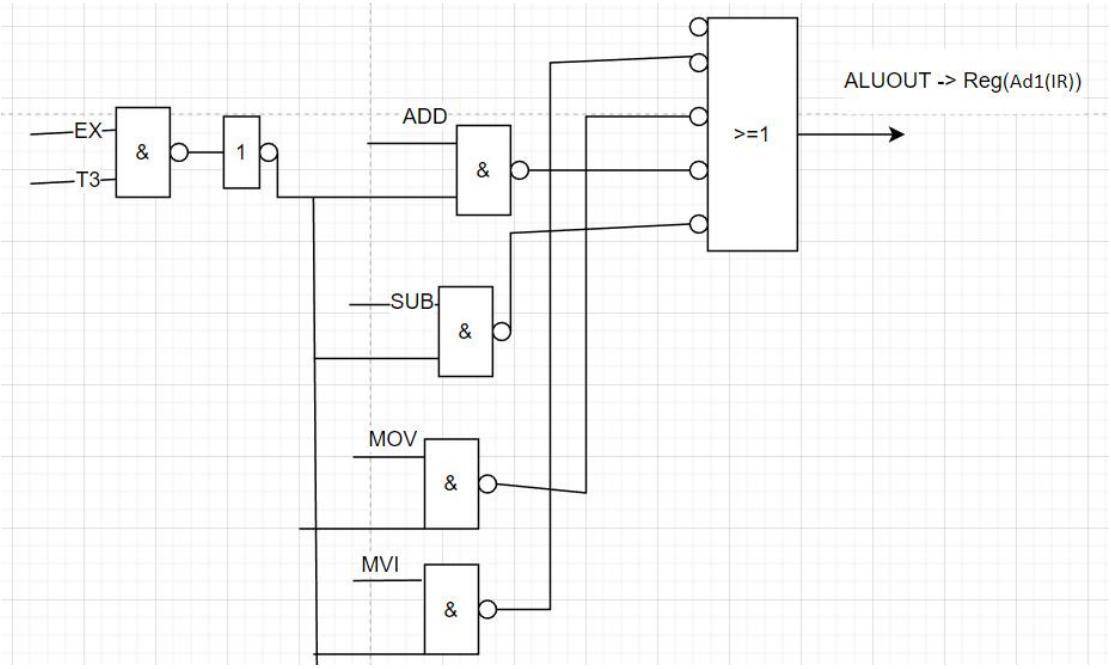
控制信号:

ALUOUT -> Reg(Ad1(IR))

微操作命令的最简表达式:

=EX*T3 *(ADD+SUB+MOV+MVI)

控制信号逻辑图:



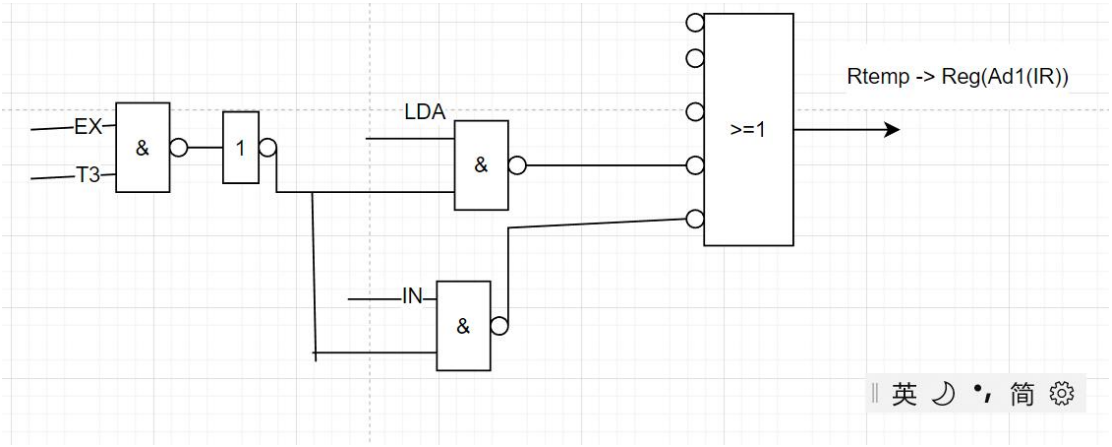
控制信号:

Rtemp -> Reg(Ad1(IR))

微操作命令的最简表达式:

=EX*T3 *(LDA+IN)

控制信号逻辑图:



控制信号:

If ALUOUT==0

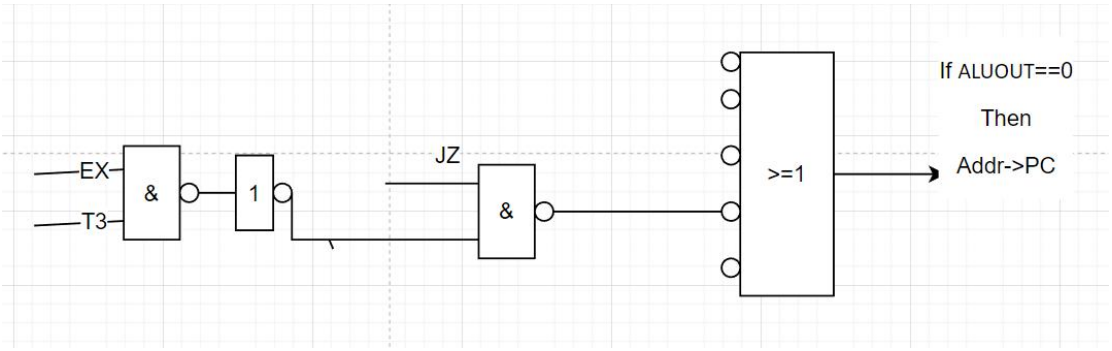
Then

Addr->PC

微操作命令的最简表达式:

=EX*T3 *(JZ)

控制信号逻辑图:



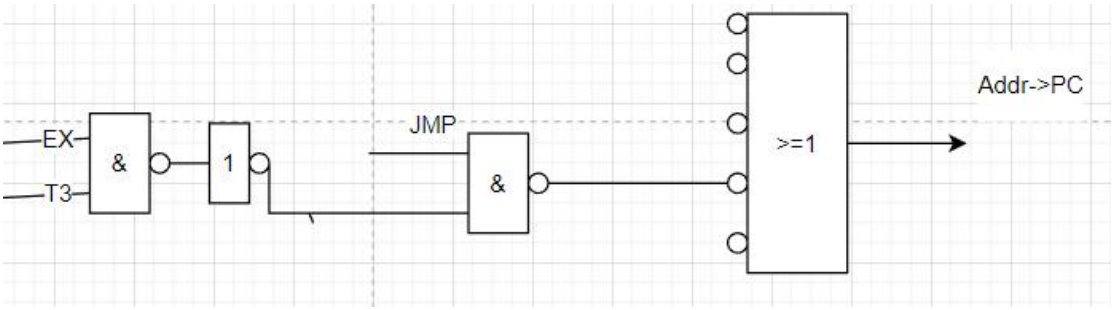
控制信号:

Addr->PC

微操作命令的最简表达式:

=EX*T3 *(JMP)

控制信号逻辑图:



六、加分项 2：用 Verilog 实现该 CPU，并仿真验证其功能。

（一）设计文件，模块介绍

1．CPU(CPU.v)

将取指，译码，执行，访存，写回模块集中管理一起调用，组成一个完成的可使用的 CPU

2．CPU 接口管理模块（CPU_export）

1.负责 CPU 对外界的读写交互的管理，在 clk 上升沿时期可以根据 RW，en，MemIO 来对主存储器或者 IO 接口进行读写
2.模拟着主存储器和 IO 接口数据的内容，模拟了十条主存储器地址从 0000_0000_0000_0000 到 0000_0000_0000_1001 的内容,模拟了 IO 接口数据地址为 0000_0000_1111_1111

3．取指阶段（取指模块 Fetch）：

T0 周期：RW 设为 0 代表 R，使能端 en 设为 1 有效，MemIO 设为 0 代表 Mem，向外输出 PC_out,通过 PC_out 来向 CPU 接口（调用 CPU_export 模块）给出指令地址，由 CPU 接口与主存储器进行读写，然后返回 IR 给 CPU 接口，再返回 IR 给 fetch 模块
T1 周期：将 IR 输出给其他模块
执行阶段（Decode 模块）
模块中保存了寄存器文件，管理着 R0~R7

4．执行阶段(Decode 模块)

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

5．执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

6．执行阶段(Access 模块)

T2 周期：根据输入的 IR，valA，valB,X，Addr，根据不同的指令，如果是需要读写主存储器或者 IO 接口的，会通过 CPU 接口（调用 CPU_export 模块）设置 RW 读写位，en 使能端，MemIO 主存储器或接口端，为相应的数值，如果是读则读取的值会在 data_read,如果是写则将写入的值在 data_write。

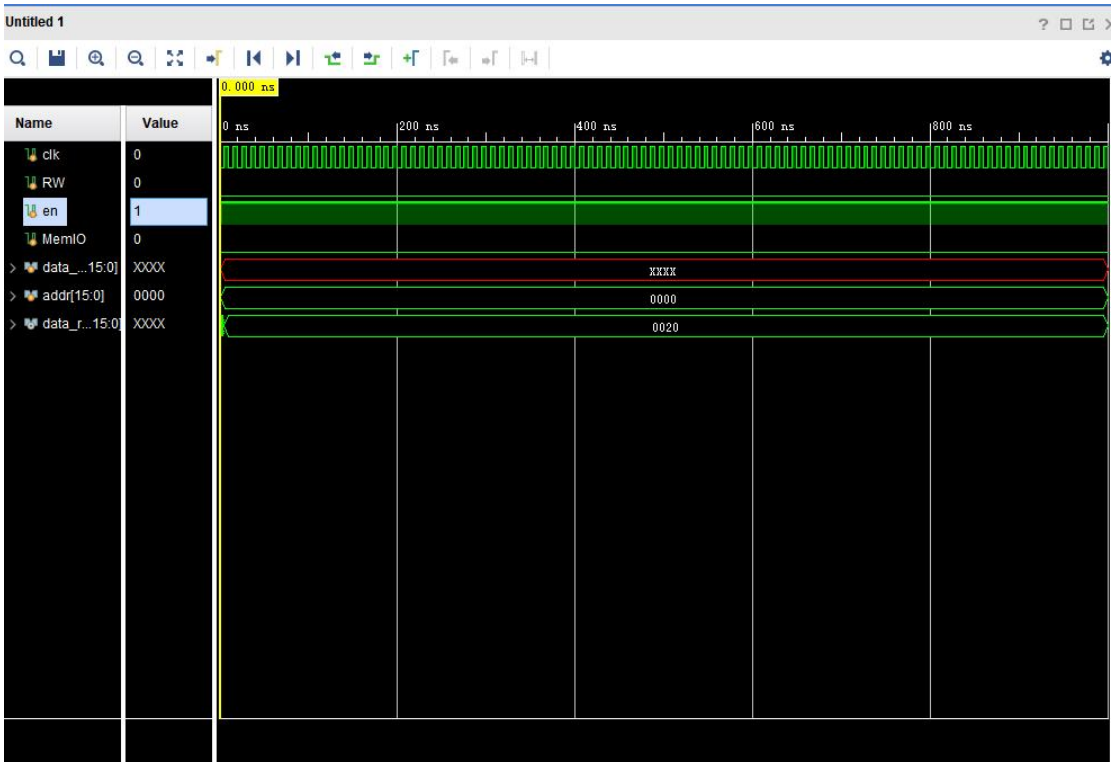
7．执行阶段(WriteBack 模块）

T3 周期：根据输入的 IR,ALUOUT,Rtemp,Addr,T3，根据不同的指令将需要更新的数值更新对应的寄存器，则会输出 R_select 写入的寄存器的编号,R_data 写入的寄存器的值,R_en 写入的使能端
如果是需要 JUMP 之类的跳转指令修改 PC，则会输出 PC_jump_en 跳转指令更新 PC 的使能端,PC_jump_data 跳转指令更新 PC 的新 PC 值

（二）仿真文件介绍，分析

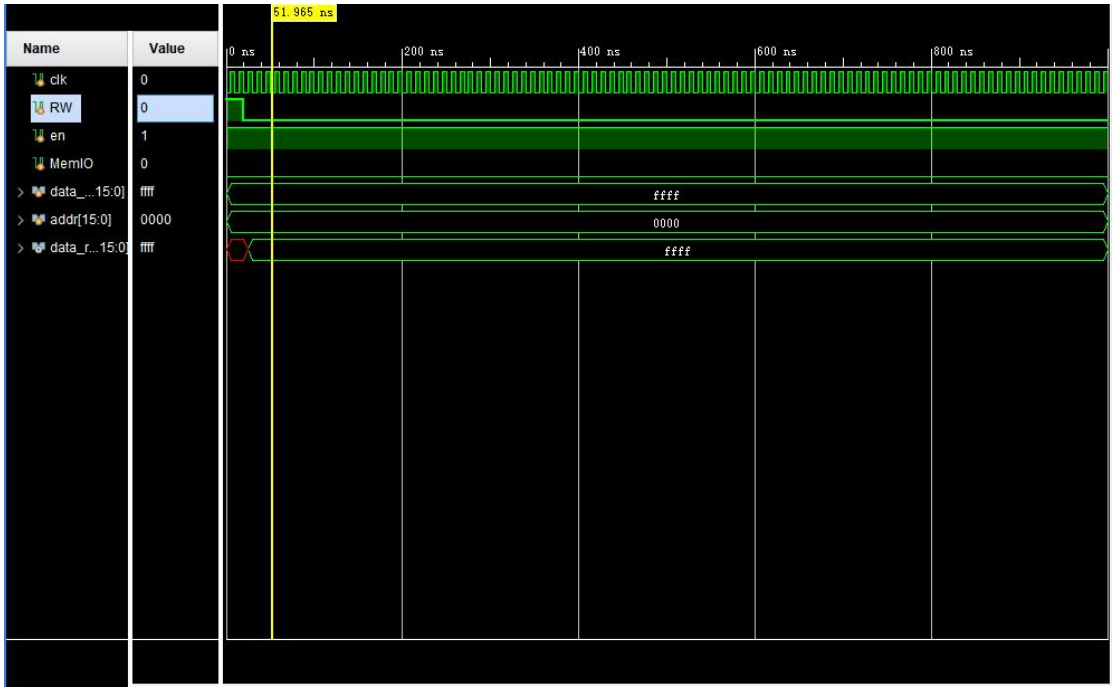
1．对每个模块进行单独的仿真测试

CPU_export_read



测试 CPU 接口模块
输入 addr 0000_0000_0000_0000
Data_write 为空，所以应该显示 XXXX
取出该地址指令（data_read）00000_000_001_00000
成功

CPU_export_write



测试 CPU 接口模块，

输入 addr 0000_0000_0000_0000

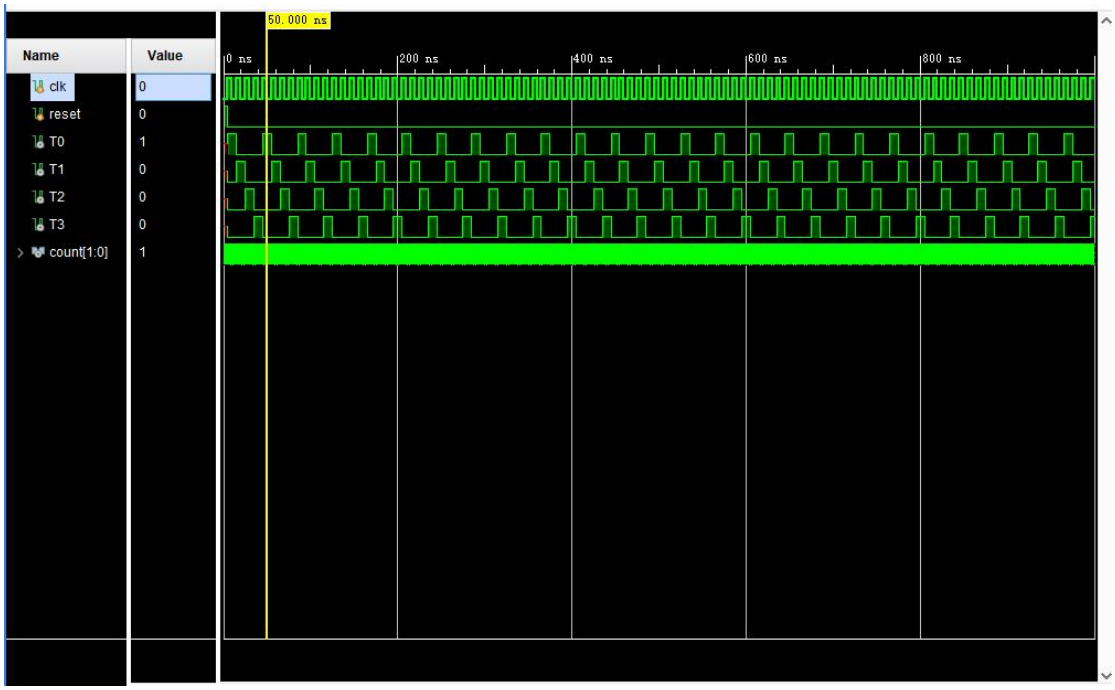
向该地址写入 data_write 11111_111_111_11111

然后查看 addr 0000_0000_0000_0000

发现已经成为 data_read 11111_111_111_11111

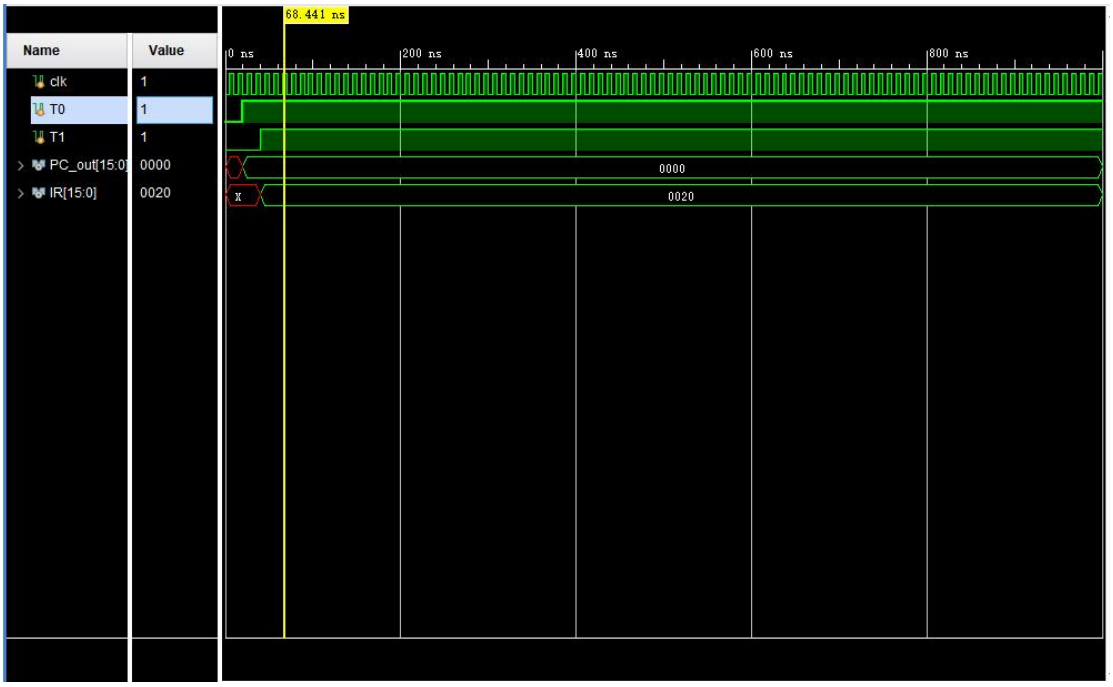
成功

Clock_tb



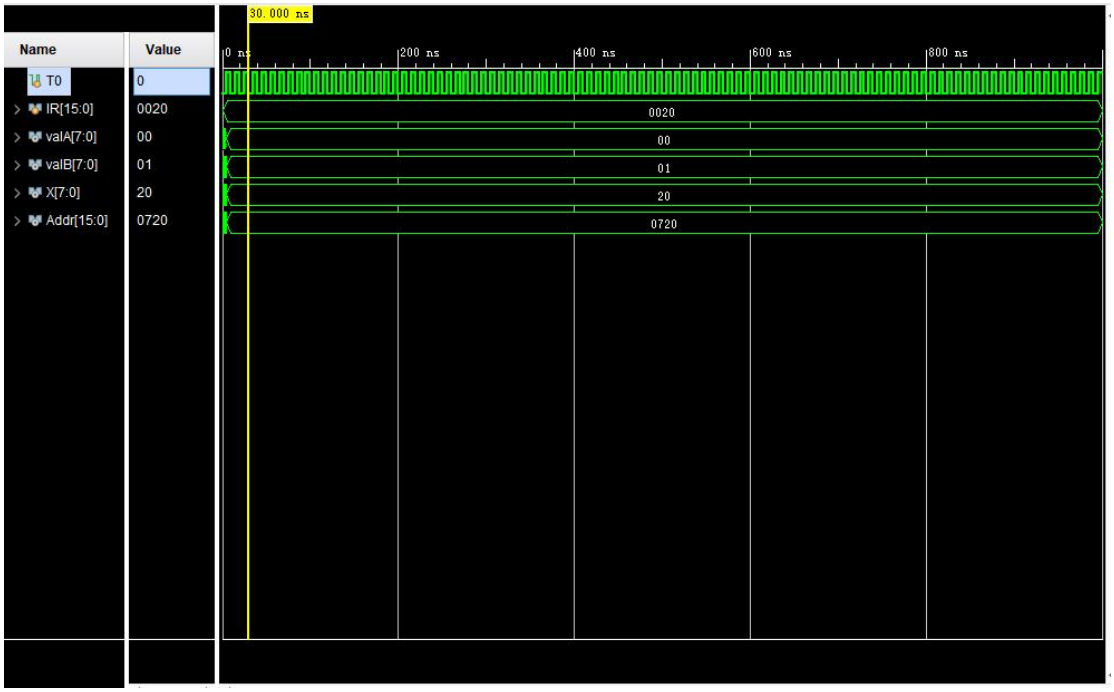
测试 Clock 模块
让其产生周期 T0~T3
成功

Fetch_tb



测试取指模块
PC 为默认 16'b0000_0000_0000_0000
取出 IR 指令 00000_000_001_00000
成功

Decode_tb



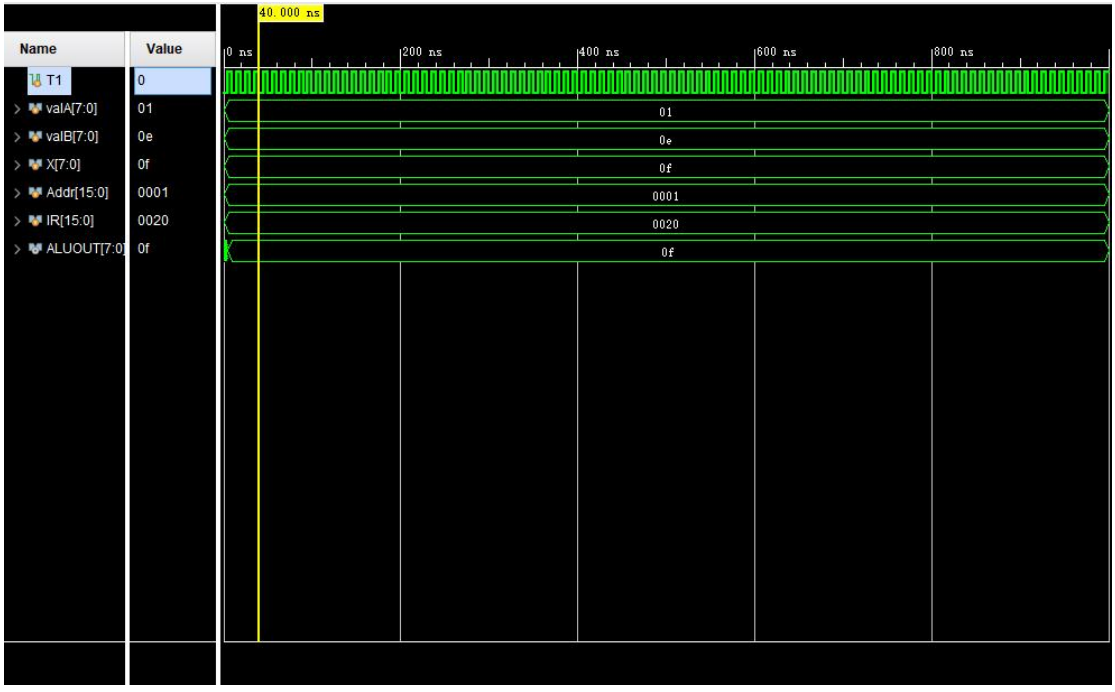
测试译码模块，

输入 IR 00000_000_001_00000，

译码得到输出 valA，valB，X，Addr

成功

Execute_tb



测试执行模块，

输入 IR 00000_000_001_00000，valA，valB，X，Addr

译码得到输出 ALUOUT
成功

2．整体测试中，十条指令样例给定的数据环境

主存储器的内容
主存中存的 PC 地址和指令，一共模拟十条，对应着十条不同的指令

地址	数据	数据含义说明
0000_0000_0000_0000	00000_000_001_00000	ADD R0+R1
0000_0000_0000_0001	00001_011_001_00000	SUB R3-R1
0000_0000_0000_0010	00010_000_001_00000	MOV (Rj)R1 -> R0(Ri)
0000_0000_0000_0011	00011_000_111_11111	MVI X(1111_1111) -> Ri(R0)
0000_0000_0000_0100	00100_000_0000_1000	STA R0 -> [R7//X] (X=0000_1000)
0000_0000_0000_0101	00101_000_0000_1000	LDA [R7//X] -> Ri(R0) (X=0000_1000)
0000_0000_0000_0110	00110_000_0000_1000	JZ if(Ri Ro == 0) then [R7//X]->PC (X=0000_1000)
0000_0000_0000_0111	00111_000_0000_1000	JMP [R7//X] -> PC (X=0000_1000)
0000_0000_0000_1000	01000_000_1111_1111	IN [PORT] -> Ri(R0) (PORT=1111_1111)后续在访存模块会补充为 0000_0000_1111_1111
0000_0000_0000_1001	01001_000_1111_1111	OUT Ri(R0) ->[PORT] (PORT=1111_1111)后续在访存模块会补充为 0000_0000_1111_1111
0000_0111_0000_1000	1111_1111_1111_1111	[R7//X]会涉及到的地址

IO 接口的内容

地址	数据	数据含义说明
0000_0000_1111_1111	1111_1111_1111_1111	设定这里为 port 地址，以及 port 数据

CPU 的 R0~R7 原来存储的值

R0	0000_0000;
R1	0000_0001;
R2	0000_0010;
R3	0000_0011;
R4	0000_0100;
R5	0000_0101;
R6	0000_0110;
R7	0000_0111;

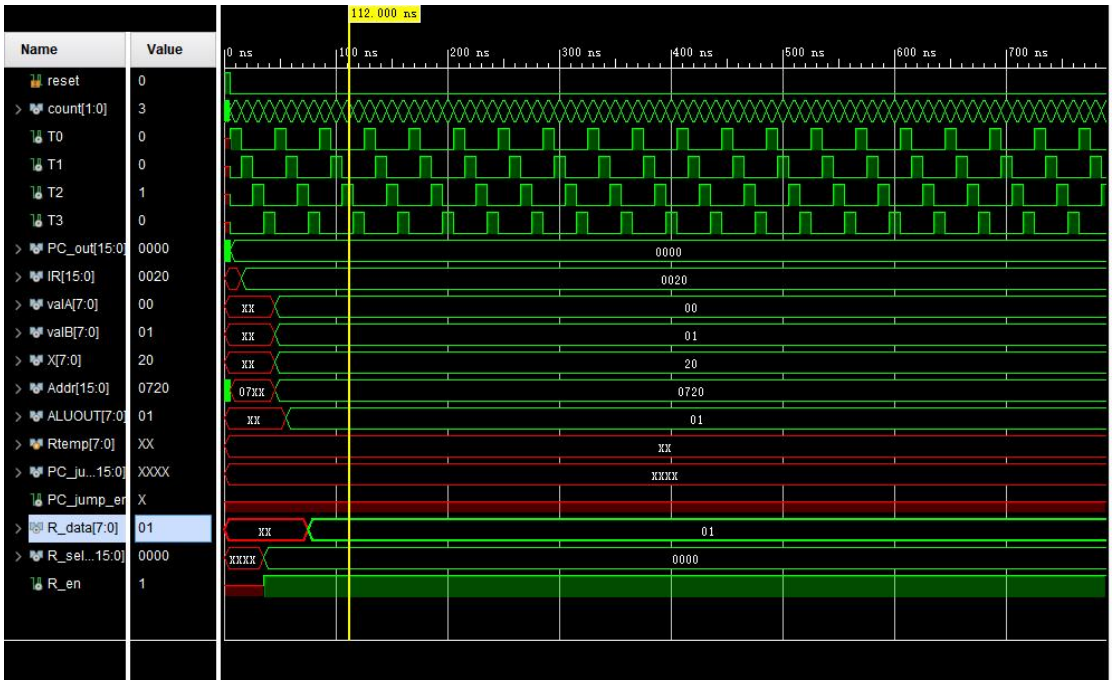
每个模块单独测试本模块功能：

启动测试方式：

通过修改 CPU 中 Fetch 当前存储的 PC 的值来选定当前一次流程所执行的指令

然后启动仿真文件 CPU_tb1.v

CPU 整体测试 ADD 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0000	00000_000_001_00000	ADD R0+R1

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out:　0000_0000_0000_0000(0000H)

取出 IR:　00000_000_001_00000(0020H)

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

通过译码得到

valA = Ri = IR[10:8] = :00h

valB = Rj = IR[7:5] = :01h

X = IR[7:0]　=:20h

Addr = [R7//X] = :0720h

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

这里因为是 ADD 操作码所以执行 ALUOUT = valA+valB = 01h

执行阶段(Access 模块)

T2 周期：

无

Rtemp 没有使用，应该为:XX

执行阶段(WriteBack 模块）

T3 周期：

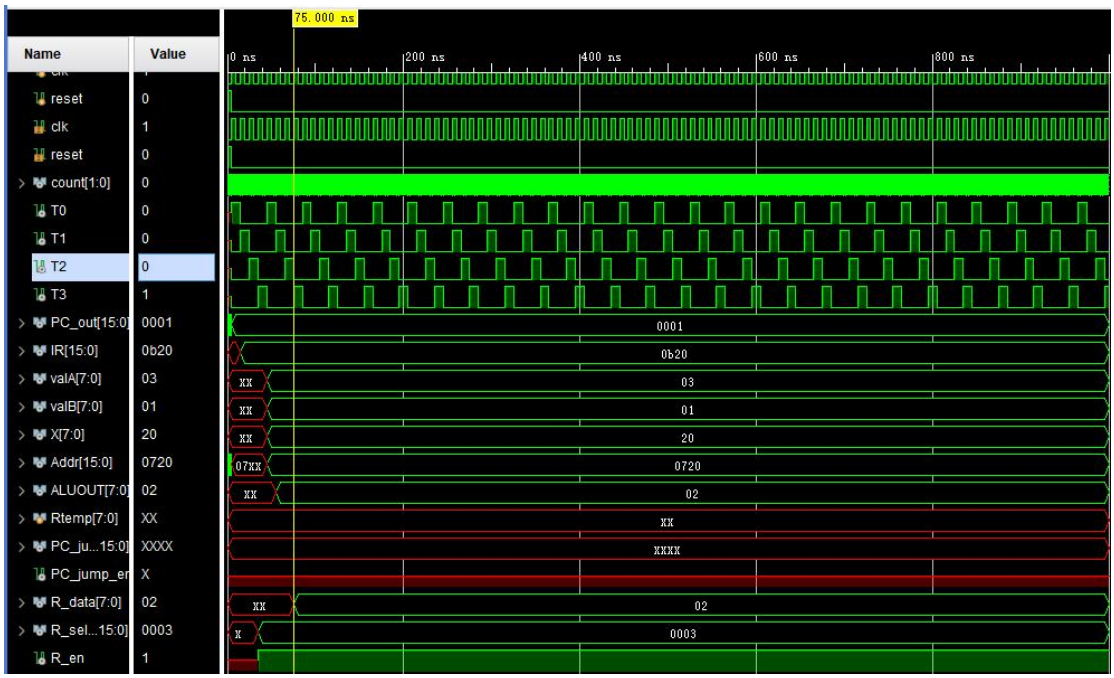
发出信号使得将 ALUOUT 的值可以写回寄存器文件修改 R0:

R_data：01h

R_select:0000h

R_en:1h

CPU 整体测试 SUB 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0001	00001_011_001_00000	SUB R3-R1

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out:　0000_0000_0000_0001

取出 IR:　00001_011_001_00000

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

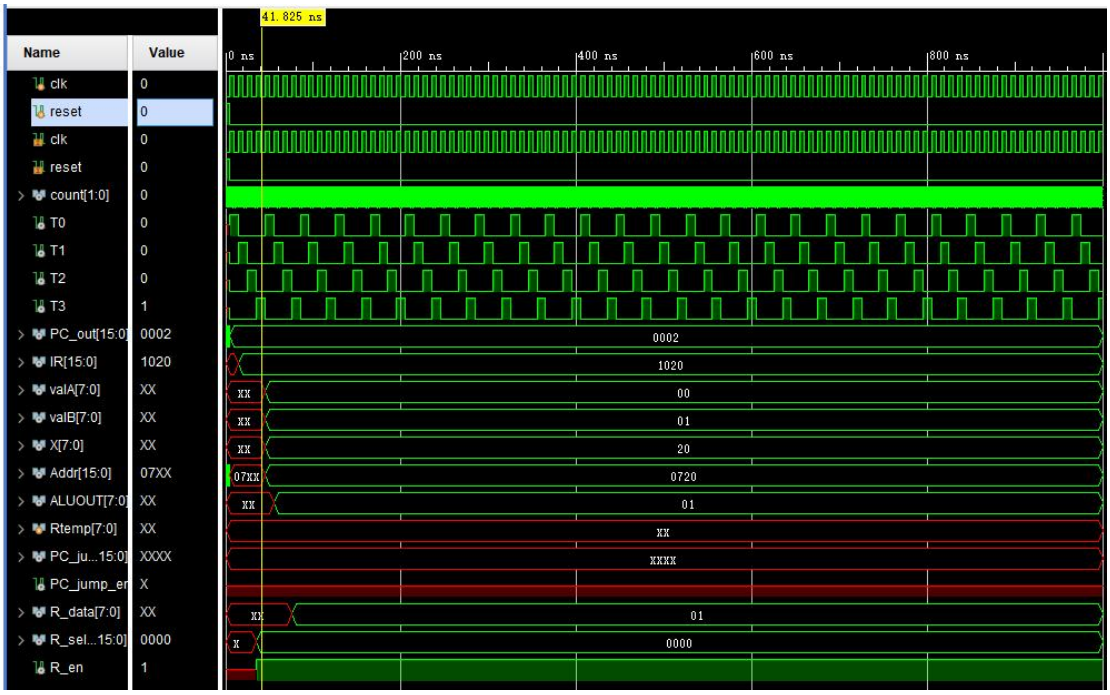
valA　= Ri = IR[10:8] = :03h

valB = Rj = IR[7:5] = :01h

X　= IR[7:0]　=:20h
Addr　= [R7//X] = :0720h

执行阶段(Execute 模块)
T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块
这里执行的是 SUB 操作所以
ALUOUT = valA-valB = 02h
执行阶段(Access 模块)
T2 周期：
无
Rtemp 没有使用，应该为:XX
执行阶段(WriteBack 模块）
T3 周期：
发出信号，使得运算结果 ALUOUT 最后写回 Ri，
可以写回寄存器文件修改 R3:
R_data=ALUOUT=02h
R_select=IR[10:8]=0003h
R_en:1h

CPU 整体测试 MOV 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0010	00010_000_001_00000	MOV (Rj)R1 -> R0(Ri)

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out:　0000_0000_0000_0010

取出 IR:　00010_000_001_00000

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA:00h

valB:01h

X:20h

Addr:0720h

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = valB = 01h

执行阶段(Access 模块)

T2 周期：

无

Rtemp 没有使用，应该为:XX

执行阶段(WriteBack 模块）

T3 周期：

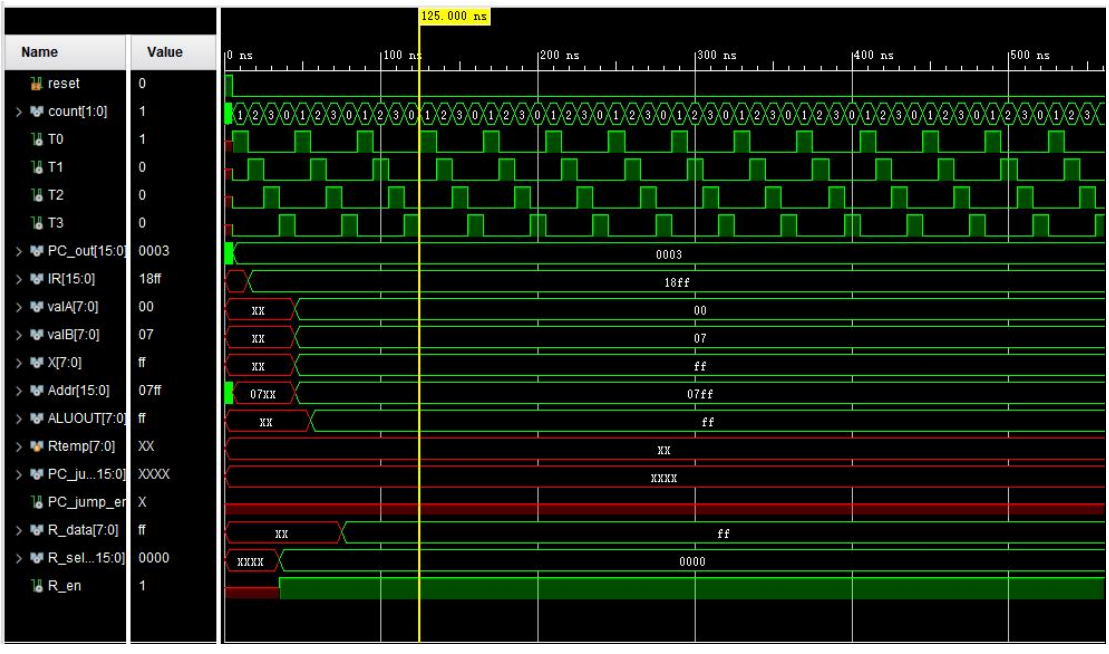
发出信号可以写回寄存器文件修改 R0

R_data=ALUOUT=01h

R_select=IR[10:8]=0000h

R_en:1h

CPU 整体测试 MVI 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0011	00011_000_111_11111	MVI X(1111_1111) -> Ri(R0)

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out:　0000_0000_0000_0011

取出 IR:　00011_000_111_11111

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA:00h

valB:07h

X:ffh

Addr:07ffh

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = X =ffh

执行阶段(Access 模块)

T2 周期：

无

Rtemp 没有使用，应该为:XX

执行阶段(WriteBack 模块）

T3 周期：

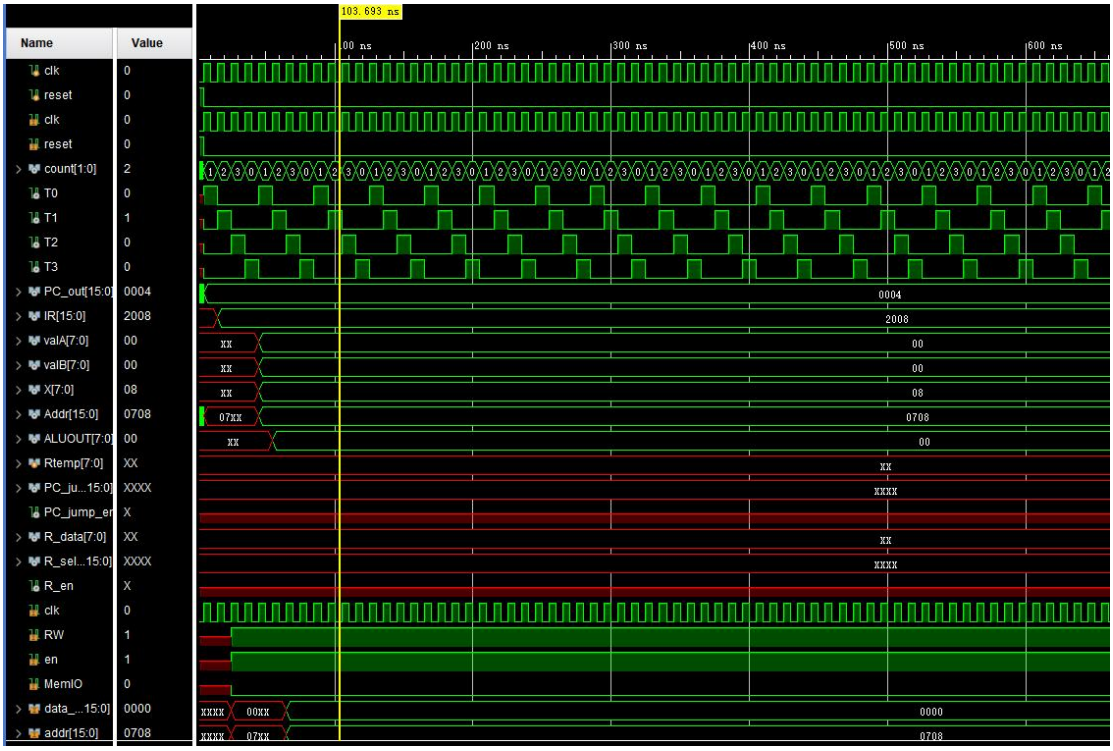
发出信号可以写回寄存器文件修改 R0

R_data=ALUOUT=ffh

R_select=IR[10:8]=0000h

R_en:1h

CPU 整体测试 STA 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0100	00100_000_0000_0000	STA R0 -> [R7//X] (X=0000_1000)

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out: 0000_0000_0000_0100

取出 IR: 00100_000_0000_0000

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA = Ri = :00h

valB = Rj = :00h

X:08h

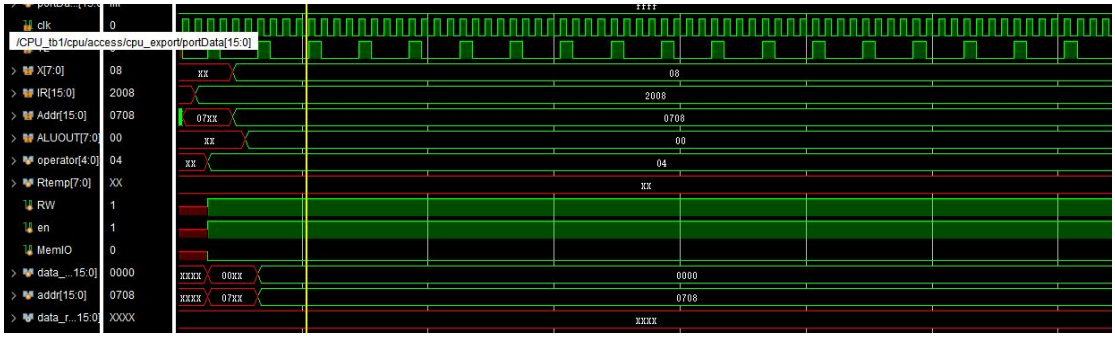
Addr=[R7//X] = :0708h

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = X =08h

执行阶段(Access 模块)



T2 周期：

RW = 1;

MemIO = 0;

en=1;

addr = Addr = 0708h;

data_write = ALUOUT = 0000h;

Data_read = XXXX

代表发出信号向 0708h 地址写入 0000h 数据

然后调用 CPU_export 模块写入主存储器

发现在 T2 时刻之后

地址为 0708h 的主存储器单元

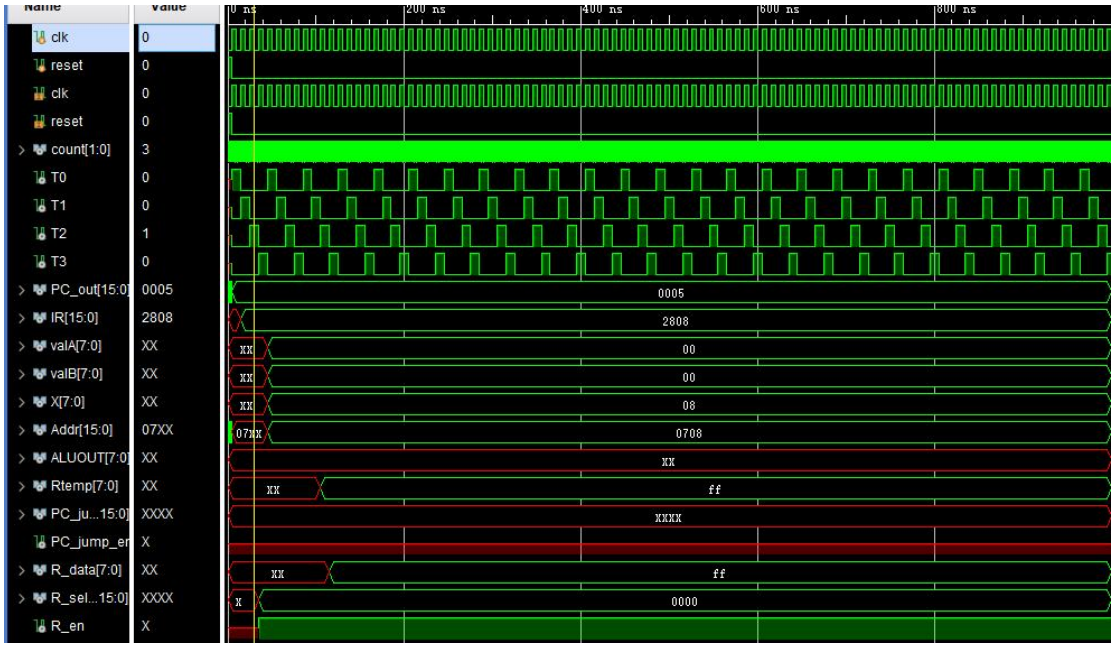
数据修改为了 0000

执行阶段(WriteBack 模块）

T3 周期：

无

CPU 整体测试 LDA 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0101	00101_000_0000_1000	LDA [R7//X] -> Ri(R0) (X=0000_1000)

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out: 0000_0000_0000_0101

取出 IR: 00101_000_0000_1000

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA = Ri = :00h

valB = Rj = :00h

X:08h

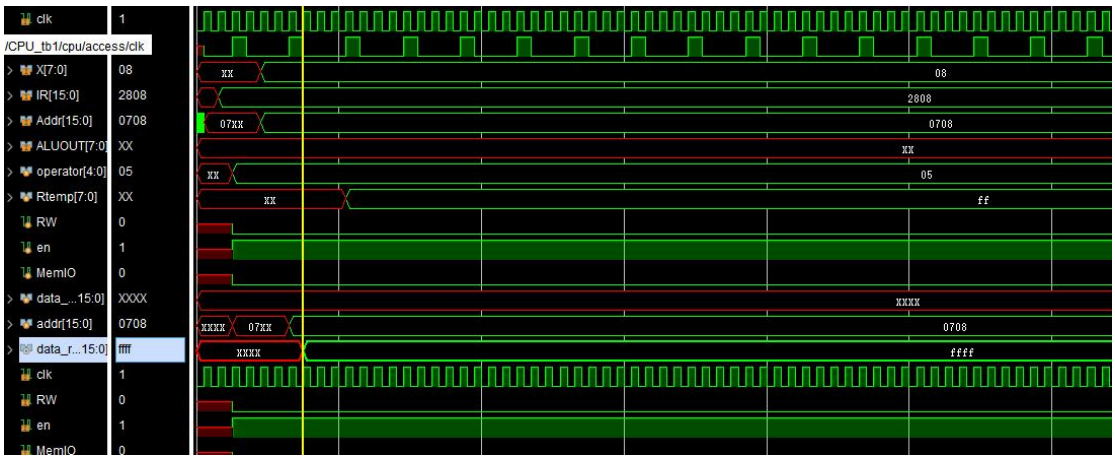
Addr=[R7//X] = :0708h

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = X =08h

执行阶段(Access 模块)



T2 周期：

RW = 0;

MemIO = 0;

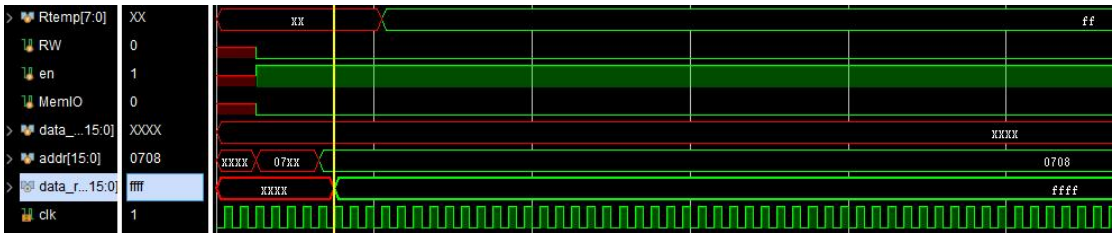
en=1;

addr = Addr = 0708h;

data_write = ALUOUT = XXXX;

代表发出信号向 0708h 地址读入数据

然后调用 CPU_export 模块读入主存储器

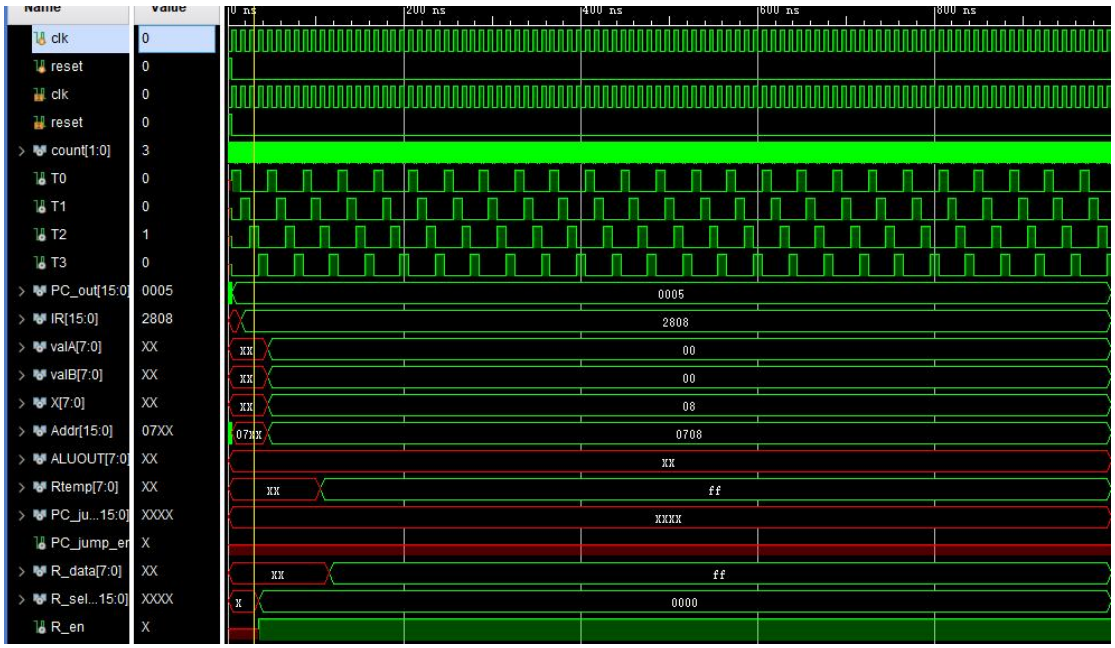


发现在 T2 时刻之后

Data_read 从 XXXX 读到 ffffh

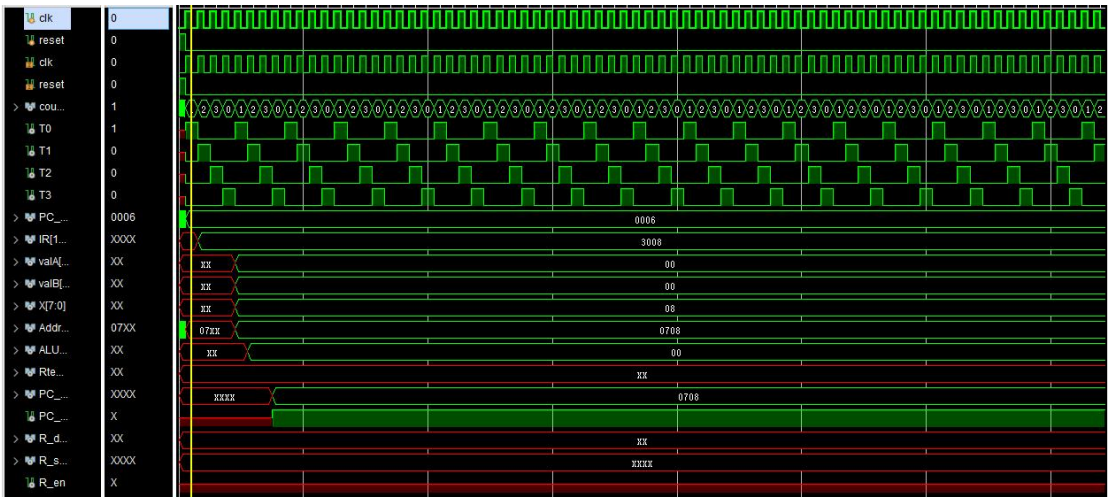
Rtemp = ffh

执行阶段(WriteBack 模块）



T3 周期：
发出信号准备修改 R
Rtemp 得到了 ffh
然后发出信号 Rdata = ffh
R-select = 000h
R_en=1h
可以向 R0 写回数据

CPU 整体测试 JZ 指令



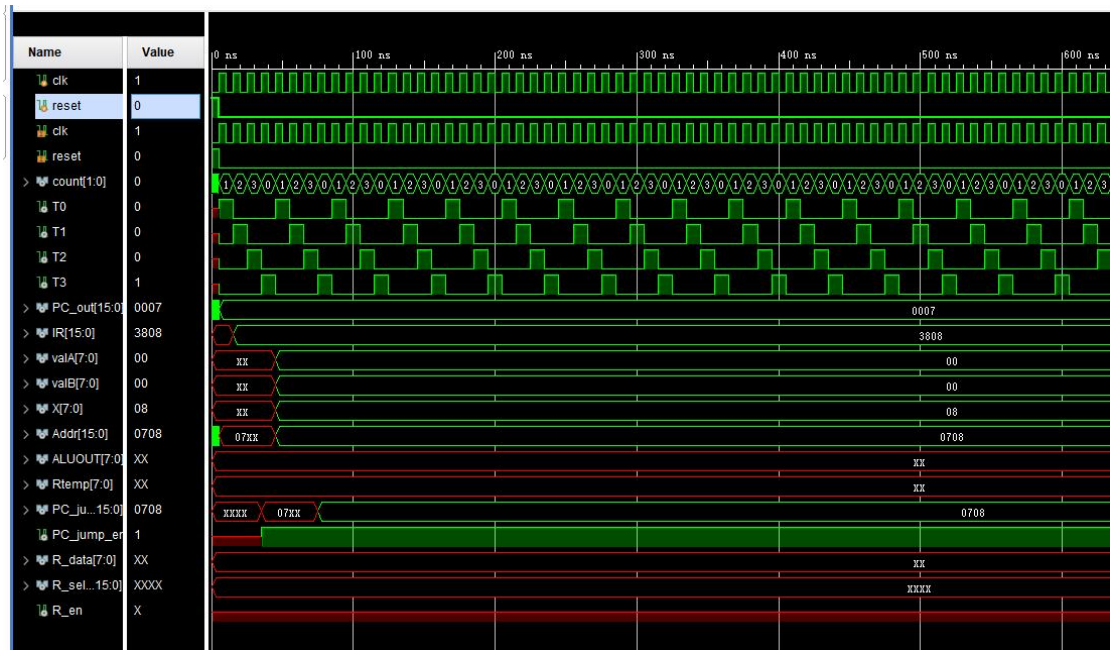
说明：

地址	数据	数据含义说明
0000_0000_0000_0110	00110_000_0000_1000	JZ if(Ri Ro == 0) then [R7//X]->PC (X=0000_1000)

取指阶段（取指模块 Fetch）：
T0 周期：根据 PC 取出 IR
根据地址 PC_out: 0000_0000_0000_0110
取出 IR: 00110_000_0000_1000
T1 周期：将 IR 输出给其他模块
执行阶段（Decode 模块）
T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块
valA:00h
valB:00h
X:08h
Addr:0708h
执行阶段(Execute 模块)
T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块
ALUOUT = valB = 00h
执行阶段(Access 模块)

T2 周期：
无
Rtemp 没有使用，应该为:XX
执行阶段(WriteBack 模块）
T3 周期：
对比 ALUOUT 即 R7==0
确认需要跳转
发出信号需要执行跳转指令
PC_jump-data=0708h
PC_jump_en=1h
会发回 PC 修改 PC 的值为 0708h

CPU 整体测试 JMP 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_0111	00111_000_0000_1000	JMP [R7//X] -> PC (X=0000_1000)

取指阶段（取指模块 Fetch）：
T0 周期：根据 PC 取出 IR
根据地址 PC_out: 0000_0000_0000_0111
取出 IR: 000111_000_0000_1000
T1 周期：将 IR 输出给其他模块
执行阶段（Decode 模块）
T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块
valA:00h

valB:00h

X:08h

Addr:0708h

执行阶段(Execute 模块)

T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = valB = 00h

执行阶段(Access 模块)

T2 周期：

无

Rtemp 没有使用，应该为:XX

执行阶段(WriteBack 模块）

T3 周期：

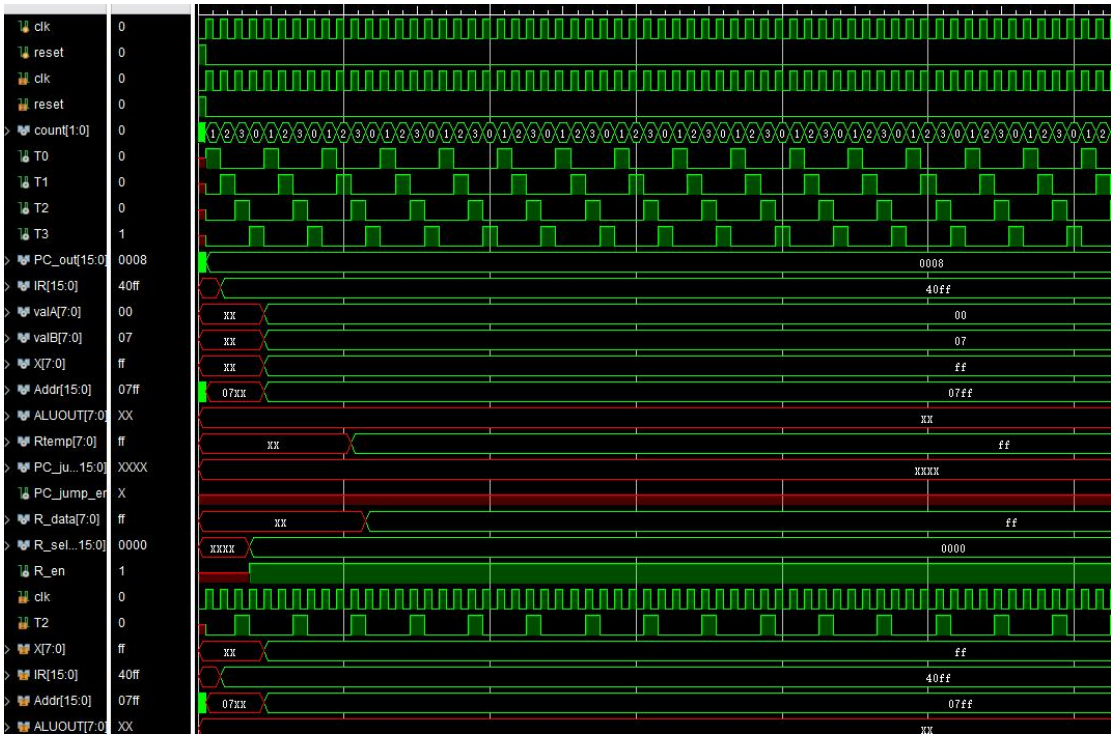
发出信号需要执行跳转指令

PC_jump-data=0708h

PC_jump_en=1h

会发回 PC 修改 PC 的值为 0708h

CPU 整体测试 IN 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_1000	01000_000_1111_1111	IN [PORT] -> Ri(R0) (PORT=1111_1111)后续在访存模块会补充为 0000_0000_1111_1111

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out:　0000_0000_0000_0101

取出 IR:　00101_000_0000_1000

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA = Ri = :00h

valB = Rj = :07h

X:ffh

Addr=[R7//X] = :07ffh

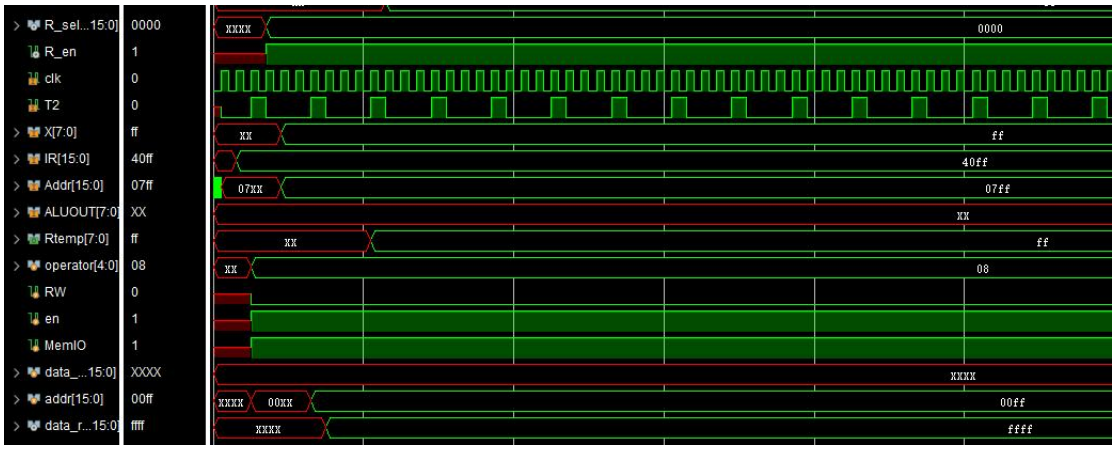
执行阶段(Execute 模块)

T1 周期：

无

所以显示时候 ALUOUT = XXXX

执行阶段(Access 模块)



T2 周期：

需要执行命令

M(X)->Rtemp

1 -> R

1 -> IO

则设置

RW = 0;

MemIO = 1;

en=1;

addr = Addr = 07ffh;

data_write = ALUOUT = XXXX;

代表发出信号向 07ffh 地址读入数据

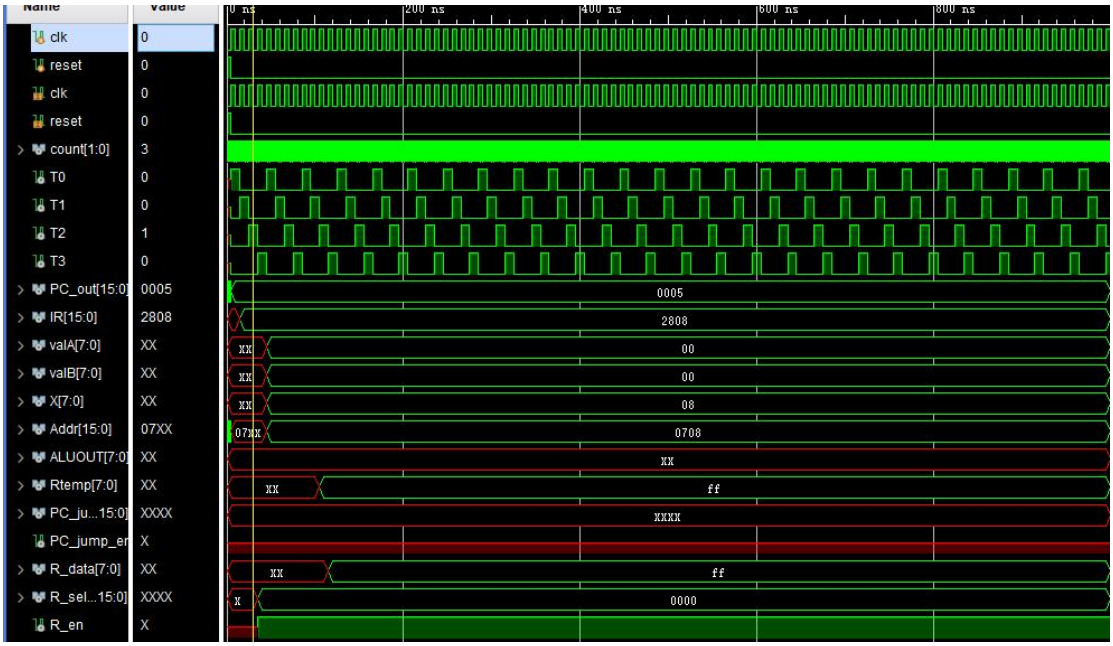
然后调用 CPU_export 模块读入 IO 接口

发现在 T2 时刻之后

Data_read 从 XXXX 读到 ffffh

Rtemp = ffh

执行阶段(WriteBack 模块）



T3 周期：

发出信号准备修改 R

Rtemp 得到了 ffh

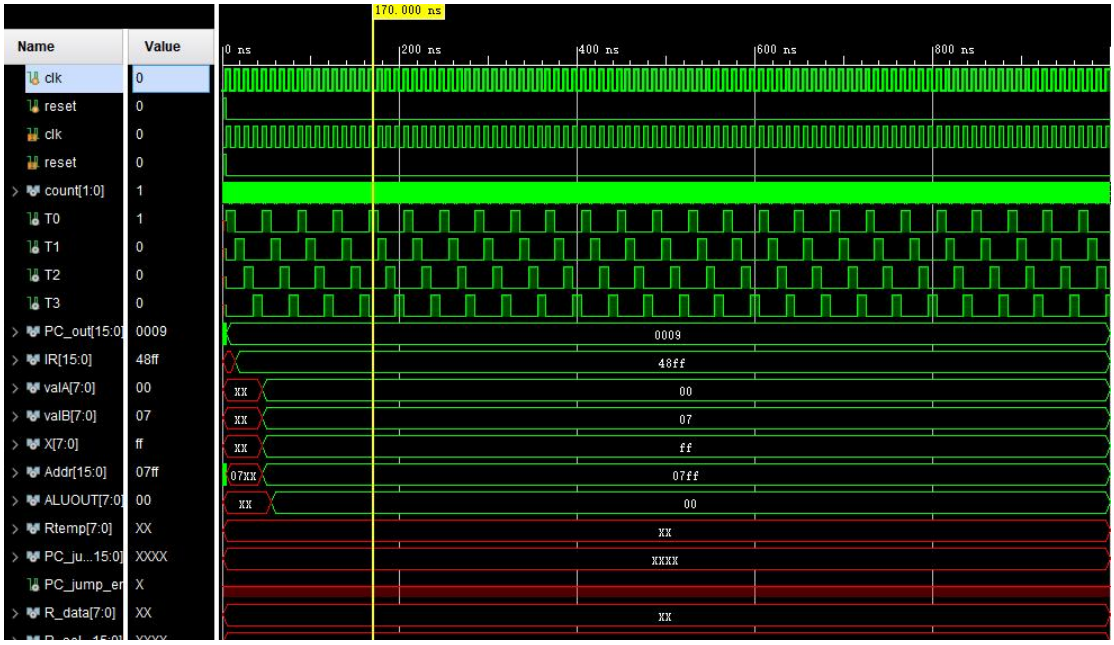
然后发出信号 Rdata = ffh

S-select = 000h

R_en=1h

可以向 R0 写回数据

CPU 整体测试 OUT 指令



说明：

地址	数据	数据含义说明
0000_0000_0000_1001	01001_000_1111_1111	OUT Ri(R0) ->[PORT] (PORT=1111_1111)后续在访存模块会补充为 0000_0000_1111_1111

取指阶段（取指模块 Fetch）：

T0 周期：根据 PC 取出 IR

根据地址 PC_out: 0000_0000_0000_1001

取出 IR: 01001_000_1111_1111

T1 周期：将 IR 输出给其他模块

执行阶段（Decode 模块）

T0 周期：根据输入的 IR，去解析对应的 Ri,Rj,X,[R7//X]的值，分别输出为 valA,valB,X,Addr 给其他模块

valA = Ri = :00h

valB = Rj = :07h

X:ffh

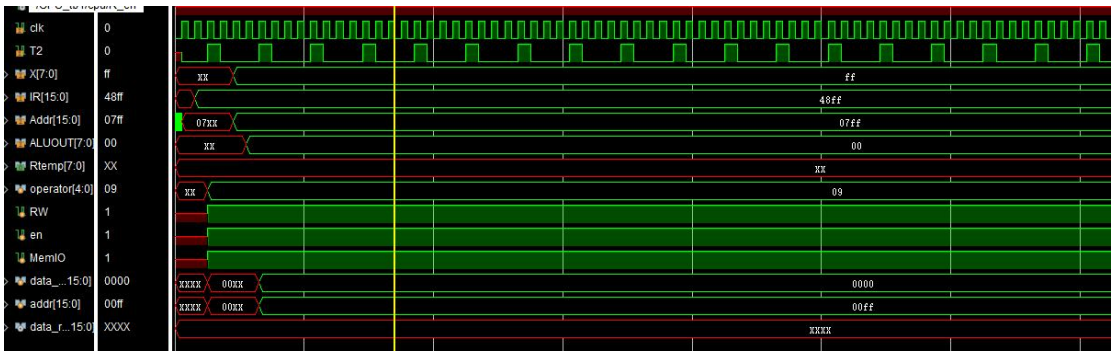
Addr=[R7//X] = :07ffh

执行阶段(Execute 模块)

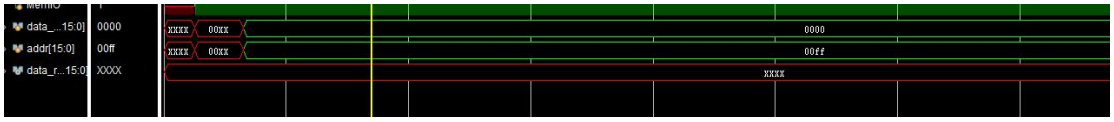
T1 周期：根据输入的 IR，valA，valB,X，Addr 做特定的操作，给 ALUOUT 附上特定的值，然后输出 ALUOUT 更新后的值给其他模块

ALUOUT = X =ffh

执行阶段(Access 模块)



T2 周期：
要执行指令：
ALUOUT ->M(X)
2->W
1 -> IO
则设定
RW = 1;
MemIO = 1;
en=1;



addr = Addr = 07ffh;
data_write = ALUOUT = 0000h;
代表发出信号向 07ffh 地址写入数据
然后调用 CPU_export 模块读入 IO 接口
发现在 T2 时刻之后成功修改 IO 接口的数据
将 07ffh 地址中的 IO 数据继续改为了 0000h
执行阶段(WriteBack 模块）
T3 周期：
无

（三）verilog 源代码

1. CPU.v

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2022/11/07 16:51:42
```

```
// Design Name:
// Module Name: CPU
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module CPU(
input wire clk,
input wire reset
);

wire [1:0] count;
wire T0;
wire T1;
wire T2;
wire T3;
wire [15:0] PC_out;//往外输出的 PC
wire [15:0] IR;
wire [7:0] valA;
wire [7:0] valB;
wire [7:0] X;
wire [15:0] Addr;
wire [7:0] ALUOUT;
wire [7:0] Rtemp;


wire [15:0] PC_jump_data;
wire  PC_jump_en;
wire [7:0] R_data;
wire [15:0] R_select;
wire  R_en;

Clock clock(
    .clk(clk),
    .reset(reset),
    .count(count),
```

```
    .T0(T0),
    .T1(T1),
    .T2(T2),
    .T3(T3)
);

Fetch fetch(
    .clk(clk),
    .T0(T0),
    .T1(T1),
    .PC_out(PC_out),
    .IR(IR)
);
Decode decode(
    .T0(T0),
    .IR(IR),
    .valA(valA),
    .valB(valB),
    .X(X),
    .Addr(Addr)
);
Execute execute(
    .T1(T1),
    .valA(valA),
    .valB(valB),
    .X(X),
    .Addr(Addr),
    .IR(IR),
    .ALUOUT(ALUOUT)
);

Access access(
    .clk(clk),
    .T2(T2),
    .X(X),
    .IR(IR),
    .Addr(Addr),
    .ALUOUT(ALUOUT),
    .Rtemp(Rtemp)
);
WriteBack writeBack(
    .T3(T3),
    .Rtemp(Rtemp),
    .ALUOUT(ALUOUT),
```

```
    .Addr(Addr),
    .IR(IR),
    .PC_jump_data(PC_jump_data),
    .PC_jump_en(PC_jump_en),
    .R_data(R_data),
     .R_select(R_select),
     .R_en(R_en)
);

endmodule
```

2. CPU_export.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/08 18:26:04
// Design Name:
// Module Name: CPU_export
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module CPU_export(
    input clk,
    input wire RW,      //0->R,1->W
```

```
26.     input wire en,      //en =1 代表可以
27.     input wire MemIO,//这次是访问 IO 还是 Mem,0->mem,1->IO
28.     input wire [15:0] data_write,
29.     input wire [15:0] addr,
30.     output reg [15:0] data_read
31.     );
32.     //模拟存储器
33.     reg [15:0] addr0;
34.     reg [15:0] data0;
35.     reg [15:0] addr1;
36.     reg [15:0] data1;
37.     reg [15:0] addr2;
38.     reg [15:0] data2;
39.     reg [15:0] addr3;
40.     reg [15:0] data3;
41.     reg [15:0] addr4;
42.     reg [15:0] data4;
43.     reg [15:0] addr5;
44.     reg [15:0] data5;
45.     reg [15:0] addr6;
46.     reg [15:0] data6;
47.     reg [15:0] addr7;
48.     reg [15:0] data7;
49.     reg [15:0] addr8;
50.     reg [15:0] data8;
51.     reg [15:0] addr9;
52.     reg [15:0] data9;
53.     reg [15:0] addrR7X;
54.     reg [15:0] dataR7X;
55.     //模拟 IO 接口
56.     reg [15:0] portAddr;
57.     reg [15:0] portData;
58.     initial
59.         begin
60.             //主存中存的 PC 地址和指令
61.             addr0 = 16'b0000_0000_0000_0000;//地址
62.             data0 = 16'b00000_000_001_00000;//ADD R0+R1                  //PC 指令数据
63.
64.             addr1 = 16'b0000_0000_0000_0001;//地址
65.             data1 = 16'b00001_011_001_00000;//SUB R3-R1                  //PC 指令数据
66.
67.             addr2 = 16'b0000_0000_0000_0010;//地址
68.             data2 = 16'b00010_000_001_00000;//MOV (Rj)R1 -> R0(Ri)                  //PC
    指令数据
```

```
69.
70.             addr3 = 16'b0000_0000_0000_0011;//地址
71.             data3 = 16'b00011_000_111_11111;//MVI X(1111_1111) -> Ri(R0)                    //PC 指令数据
72.
73.             addr4 = 16'b0000_0000_0000_0100;//地址
74.             data4 = 16'b00100_000_0000_1000;//STA R0 -> [R7//X] (X=0000_1000)
                //PC 指令数据
75.
76.             addr5 = 16'b0000_0000_0000_0101;//地址
77.             data5 = 16'b00101_000_0000_1000;//LDA [R7//X] -> Ri(R0) (X=0000_1000)
                  //PC 指令数据
78.
79.             addr6 = 16'b0000_0000_0000_0110;//地址      (R0 确实为 0)
80.             data6 = 16'b00110_000_0000_1000;//JZ if(Ri Ro == 0) then [R7//X]->PC
(X=0000_1000)          //PC 指令数据
81.
82.             addr7 = 16'b0000_0000_0000_0111;//地址
83.             data7 = 16'b00111_000_0000_1000;//JMP [R7//X] -> PC  (X=0000_1000)
                //PC 指令数据
84.
85.             addr8 = 16'b0000_0000_0000_1000;//地址
86.             data8 = 16'b01000_000_1111_1111;//IN [PORT] -> Ri(R0) (PORT=1111_1111)后
续在访存模块会补充为 0000_0000_1111_1111                    //PC 指令数据
87.
88.             addr9 = 16'b0000_0000_0000_1001;//地址
89.             data9 = 16'b01001_000_1111_1111;//OUT Ri(R0) ->[PORT] (PORT=1111_1111)
后续在访存模块会补充为 0000_0000_1111_1111                    //PC 指令数据
90.
91.             //[R7//X]会涉及到的地址
92.             addrR7X = 16'b0000_0111_0000_1000;
93.             dataR7X = 16'b1111_1111_1111_1111;
94.             //IO 接口
95.             portAddr = 16'b0000_0000_1111_1111; //设定这里为 port 地址
96.             portData = 16'b1111_1111_1111_1111; //设定为 port 数据
97.         end
98.     always@(posedge clk)
99.         begin
100.            if(en == 1 && MemIO == 0)
101.            begin
102.                //从主存中读 R
103.                if(RW == 0)
104.                begin
105.                    if(addr == addr0)
```

```
106.                 data_read = data0;
107.             else if(addr == addr1)
108.                 data_read = data1;
109.             else if(addr == addr2)
110.                 data_read = data2;
111.          else if(addr == addr3)
112.                 data_read = data3;
113.             else if(addr == addr4)
114.                 data_read = data4;
115.             else if(addr == addr5)
116.                 data_read = data5;
117.             else if(addr == addr6)
118.                 data_read = data6;
119.             else if(addr == addr7)
120.                 data_read = data7;
121.             else if(addr == addr8)
122.                 data_read = data8;
123.             else if(addr == addr9)
124.                 data_read = data9;
125.             else if(addr == addrR7X)
126.                 data_read = dataR7X;
127.         end
128.
129.         //从主存中写 W
130.         if(RW == 1)
131.         begin
132.             if(addr == addr1)
133.                 data1 = data_write;
134.             else if(addr == addr2)
135.                 data2 = data_write;
136.             else if(addr == addr3)
137.                 data3 = data_write;
138.             else if(addr == addr4)
139.                 data4 = data_write;
140.             else if(addr == addr5)
141.                 data5 = data_write;
142.             else if(addr == addr6)
143.                 data6 = data_write;
144.             else if(addr == addr7)
145.                 data7 = data_write;
146.             else if(addr == addr8)
147.                 data8 = data_write;
148.             else if(addr == addr9)
149.                 data9 = data_write;
```

```
150.                else if(addr == addrR7X)
151.                    dataR7X = data_write;
152.            end
153.        end
154.
155.
156.        if(en == 1 && MemIO == 1)
157.        begin
158.            //从 IO 中读 R
159.            if(RW == 0)
160.            begin
161.                if(addr == portAddr)
162.                    data_read = portData;
163.            end
164.            //从 IO 中写 W
165.            if(RW == 1)
166.            begin
167.                if(addr == addr1)
168.                    portData = data_write;
169.            end
170.        end
171.    end
172. endmodule
```

3. Clock.v

```
1.  //N 进制计数器
2.  module Clock(
3.      input wire clk,
4.      input wire reset,
5.      output reg [1:0] count,
6.      output reg T0,
7.      output reg T1,
8.      output reg T2,
9.      output reg T3);
10.  parameter N=4;  //定义进制 N 的值
11.     initial
12.     begin
13.         count = 2'b00;
14.     end
15.  always@(posedge clk) //当有时钟上升沿来到，执行下列语句
16.     begin
17.         if(count == 2'b00)
18.         begin
```

```
            T0  <= 1'b1;
            T1  <= 1'b0;
            T2  <= 1'b0;
            T3  <= 1'b0;
        end
        if(count == 2'b01)
        begin
            T0  <= 1'b0;
            T1  <= 1'b1;
            T2  <= 1'b0;
            T3  <= 1'b0;
        end
        if(count == 2'b10)
        begin
            T0  <= 1'b0;
            T1  <= 1'b0;
            T2  <= 1'b1;
            T3  <= 1'b0;
        end
        if(count == 2'b11)
        begin
            T0  <= 1'b0;
            T1  <= 1'b0;
            T2  <= 1'b0;
            T3  <= 1'b1;
        end
    if(reset) //清零信号=1，则清零
     count<=2'b00; //有效计数器清零
    else
     if(count == 2'b11) //是否计导第 N 个数(0-N)
      count<=2'b00; //清零
     else  //否则加 1
      count<=count+1;
     end
endmodule
```

4. Fetch.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date: 2022/11/08 18:25:33
// Design Name:
// Module Name: Fetch
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Fetch(
    input clk,
    input wire T0,
    input wire T1,
    output reg [15:0] PC_out,//往外输出的 PC
    output reg [15:0] IR
    );
    reg RW;      //0->R,1->W
    reg en;     //en =1 代表可以
    reg MemIO;//这次是访问 IO 还是 Mem,0->mem,1->IO
    reg [15:0] data_write;
    reg [15:0] PC;
    wire [15:0] IR_data;//Mem 找到之后给的指令数据

    CPU_export cpu_export(
        .clk(clk),
        .RW(RW),
        .en(en),
        .MemIO(MemIO),
        .data_write(data_write),
        .addr(PC),
        .data_read(IR_data)
    );
    initial
    begin
        PC = 16'b0000_0000_0000_0001;//默认 PC
        en =1'b0;  //默认关闭
```

```
    end

    always @(posedge T0)
    begin
        RW =1'b0;
        en =1'b1;
        MemIO =1'b0;

        PC_out <= PC;
    end

    always @(posedge T1)
    begin
        IR = IR_data;
    end

endmodule
```

5. Execute.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/08 16:09:38
// Design Name:
// Module Name: Execute
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Execute(
```

```
24.     input T1,
25.     input wire [7:0] valA,
26.     input wire [7:0] valB,
27.     input wire [7:0] X,
28.     input wire [15:0] Addr,
29.     input wire [15:0] IR,
30.     output reg [7:0] ALUOUT
31.     );
32.     reg [4:0] operator;
33.     always @(posedge T1)
34.     begin
35.         operator[4:0] = IR[15:11];
36.         if(operator == 5'b00000)//ADD
37.         begin
38.             ALUOUT = valA + valB;
39.         end
40.         if(operator == 5'b00001)//SUB
41.         begin
42.             ALUOUT = valA - valB;
43.         end
44.         if(operator == 5'b00010)// MOV
45.         begin
46.             ALUOUT = valB;
47.         end
48.         if(operator == 5'b00011)//MVI
49.         begin
50.             ALUOUT = X;
51.         end
52.         if(operator == 5'b00100)//STA
53.         begin
54.             ALUOUT = valA;
55.         end
56.         if(operator == 5'b00101)//LDA
57.         begin
58.
59.         end
60.         if(operator == 5'b00110)//JZ
61.         begin
62.             ALUOUT = valA;
```

```
63.         end
64.
65.         if(operator == 5'b00111)//JMP
66.         begin
67.
68.         end
69.
70.         if(operator == 5'b01000)//IN
71.         begin
72.
73.         end
74.         if(operator == 5'b01001)//OUT
75.         begin
76.             ALUOUT = valA;
77.         end
78.     end
79.
80. endmodule
```

6. Access.v

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2022/11/08 19:53:55
7.  // Design Name:
8.  // Module Name: Access
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
```

```
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. 
22. 
23. module Access(
24.     input clk,
25.     input wire T2,
26.     input wire [7:0] X,
27.     input wire [15:0] IR,
28.     input wire [15:0] Addr,
29.     input wire [7:0] ALUOUT,
30.     output reg [7:0] Rtemp
31.     );
32.     reg [4:0] operator;
33. 
34. 
35.     reg RW;      //0->R,1->W
36.     reg en;      //en =1 代表可以
37.     reg MemIO;//这次是访问 IO 还是 Mem,0->mem,1->IO
38.     reg [15:0] data_write;
39.     reg [15:0] addr;
40.     wire [15:0] data_read;
41. 
42.     CPU_export cpu_export(
43.     .clk(clk),
44.     .RW(RW),
45.     .en(en),
46.     .MemIO(MemIO),
47.     .data_write(data_write),
48.     .addr(addr),
49.     .data_read(data_read)
50.     );
51.     always @(posedge T2)
52.         begin
53.             operator[4:0] = IR[15:11];
54.             if(operator == 5'b00100)//STA
55.             begin
56.                 //模拟向主存写回
57.                 //ALUOUT->M(Addr)
58.                 //ALUOUT->M(Addr)
59.                 //1->W
60.                 //1->Mem
```

```
61.             RW = 1;
62.             MemIO = 0;
63.             en=1;
64.             addr = Addr;
65.             data_write = ALUOUT;
66.         end
67.         if(operator == 5'b00101)//LDA
68.         begin
69.             //模拟向主存读入
70.             RW = 0;
71.             MemIO = 0;
72.             en=1;
73.             addr = Addr;
74.             Rtemp = data_read;
75.         end
76.
77.         if(operator == 5'b01000)//IN
78.         begin
79.             //从 IO 读入
80.             RW = 0;
81.             MemIO = 1;
82.             en=1;
83.             //需要拼接
84.             addr = {8'b0000_0000,X};
85.             Rtemp = data_read;
86.         end
87.         if(operator == 5'b01001)//OUT
88.         begin
89.             //从 IO 写入
90.             RW = 1;
91.             MemIO = 1;
92.             en=1;
93.             //需要拼接
94.             addr = {8'b0000_0000,X};
95.             data_write = ALUOUT;
96.         end
97.     end
98.     //访存阶段
99.
100. endmodule
```

7. WriteBack.v

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2022/11/08 19:54:07
7.  // Design Name:
8.  // Module Name: WriteBack
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21.
22.
23. module WriteBack(
24.     input wire T3,
25.     input wire [7:0] Rtemp,
26.     input wire [7:0] ALUOUT,
27.     input wire [15:0] Addr,
28.     input wire [15:0] IR,
29.     output reg [15:0] PC_jump_data,
30.     output reg  PC_jump_en,
31.     output reg [7:0] R_data,
32.     output reg [15:0] R_select,
33.     output reg  R_en
34.     );
35.     reg [4:0] operator;
36.     always @(posedge T3)
37.     begin
38.         operator[4:0] = IR[15:11];
39.         //写回阶段
40.         //执行 ALUOUT -> Reg(Ad1(IR))
```

```
41.       if(operator == 5'b00000
42.       || operator == 5'b00001
43.       || operator == 5'b00010
44.       || operator == 5'b00011)//ADD SUB MOV MVI
45.       begin
46.          R_select = IR[10:8];
47.          R_data = ALUOUT;
48.          R_en = 1;
49.       end
50.
51.       //Rtemp -> Reg(Ad1(IR))
52.       if(operator == 5'b00101 || operator == 5'b01000) //LDA IN
53.       begin
54.          R_select =  IR[10:8];
55.          R_data  = Rtemp;
56.          R_en =1;
57.       end
58.
59.       //If ALUOUT==0
60.       // Then
61.       // Addr->PC
62.       if(operator == 5'b00110) //JZ
63.       begin
64.          if(ALUOUT == 1'd0)
65.          begin
66.             PC_jump_data = Addr;
67.             PC_jump_en = 1;
68.          end
69.       end
70.
71.       //Addr->PC
72.       if(operator == 5'b00111)//JMP
73.       begin
74.          PC_jump_data = Addr;
75.          PC_jump_en = 1;
76.       end
77.
78.
79.       if(operator == 5'b01001)//OUT
80.       begin
81.
82.       end
83.    end
```

```
84. endmodule
```

8. CPU_export_read.v

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2022/11/08 19:13:50
7.  // Design Name:
8.  // Module Name: CPU_export_read
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21.
22. /**
23. 测试 CPU 接口模块，输入 addr 0000_0000_0000_0000，取出该地址指令
    00000_000_001_00000
24. 成功
25. */
26. module CPU_export_read();
27.     reg clk;
28.     reg RW;      //0->R,1->W
29.     reg en;      //en =1 代表可以
30.     reg MemIO;//这次是访问 IO 还是 Mem,0->mem,1->IO
31.     reg [15:0] data_write;
32.     reg [15:0] addr;
33.     wire [15:0] data_read;
34.     always #5 clk=~clk;//每 10ns 一次
35.     initial
36.         begin
37.             clk = 0;
```

```
            RW=0;
            en=1;
            MemIO=0;
            addr =16'b0000_0000_0000_0000;
        end
    CPU_export cpu_export_readTest(
    .clk(clk),
    .RW(RW),
    .en(en),
    .MemIO(MemIO),
    .data_write(data_write),
    .addr(addr),
    .data_read(data_read)
    );
endmodule
```

9. CPU_export_write.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/08 19:13:50
// Design Name:
// Module Name: CPU_export_read
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

/**
测试 CPU 接口模块，
输入 addr 0000_0000_0000_0000
```

```
向该地址写入 11111_111_111_11111

然后查看 addr 0000_0000_0000_0000
发现已经成为 11111_111_111_11111

成功
*/
module CPU_export_write();
    reg clk;
    reg RW;      //0->R,1->W
    reg en;      //en =1 代表可以
    reg MemIO;//这次是访问 IO 还是 Mem,0->mem,1->IO
    reg [15:0] data_write;
    reg [15:0] addr;
    wire [15:0] data_read;
    always #5 clk=~clk;//每 10ns 一次
    initial
        begin
            //先写
            clk = 0;
            RW=1;
            en=1;
            MemIO=0;
            addr =16'b0000_0000_0000_0000;
            data_write =16'b11111_111_111_11111;

            //后通过 data_read 看
            #20
            RW=0;
            en=1;
            MemIO=0;
            addr =16'b0000_0000_0000_0000;
        end
    CPU_export cpu_export_readTest(
    .clk(clk),
    .RW(RW),
    .en(en),
    .MemIO(MemIO),
    .data_write(data_write),
    .addr(addr),
    .data_read(data_read)
    );
```

```
68. endmodule
```

10. Clock_tb.v

```
1.  `timescale 1ns/1ns //时间精度为 1ns
2.  /**
3.  测试 Clock 模块
4.  让其产生周期 T0~T3
5.  成功
6.  */
7.
8.  module Clock_tb();
9.   reg clk,reset; //initial 语句块中左边必须是 reg 型
10.  wire T0,T1,T2,T3;
11.  wire [1:0] count;
12. // defparam u.N=4; //将默认的模 6 改为模 4
13.  always #5 clk=~clk; //每 5ns 翻转一次，10ns 为一周期
14.  initial
15.   begin
16.    clk=0; reset=1; //reset=1,将 count 初始化为 4'b0000，否则 count 一直处于未知态
17.    #5 reset=0; //reset=0,计数器才能正常计数
18.   end
19.   counter u(.clk(clk),.reset(reset),.count(count),.T0(T0),.T1(T1),.T2(T2),.T3(T3));
20. endmodule
```

CPU_tb1.v

```
1.  `timescale 1ns / 1ps
2.  module CPU_tb1();
3.      reg clk;
4.      reg reset;
5.      always #5 clk=~clk;//每 10ns 一次
6.      initial
7.    begin
8.     clk=0; reset=1; //reset=1,将 count 初始化为 4'b0000，否则 count 一直处于未知态
9.     #5 reset=0; //reset=0,计数器才能正常计数
10.   end
11. CPU cpu(
12.     .clk(clk),
13.     .reset(reset)
14. );
15. endmodule
```

11. Fetch_tb.v

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Company:
4.  // Engineer:
5.  //
6.  // Create Date: 2022/11/08 19:08:28
7.  // Design Name:
8.  // Module Name: Fetch_tb_add
9.  // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21.
22. /**
23. 测试取指模块
24. PC 为默认 16'b0000_0000_0000_0000
25. 取出 IR 指令 00000_000_001_00000
26. 成功
27. */
28. module Fetch_tb_readPC();
29.     reg clk;
30.     reg T0;
31.     reg T1;
32.     wire [15:0] PC_out;//往外输出的 PC
33.     wire [15:0] IR;
34.     always #5 clk=~clk;//每 10ns 一次
35.     initial
36.     begin
37.         clk=0;
38.         T0=0;
39.         T1=0;
40.         #20 T0=~T0;
41.
```

```
        #20 T1=~T1;
    end
    Fetch fetch_test_defalut(
        .clk(clk),
        .T0(T0),
        .T1(T1),
        .PC_out(PC_out),
        .IR(IR)
    );
endmodule
```

12. Decode_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/08 16:37:23
// Design Name:
// Module Name: Decode_tb_add
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

/**
测试译码模块，输入 IR 00000_000_001_00000，
译码得到输出 valA，valB，X，Addr
成功
*/
module Decode_tb_add();
    reg T0;
```

```
    reg [15:0] IR;
    wire [7:0] valA;
    wire [7:0] valB;
    wire [7:0] X;
    wire [15:0] Addr;
    always #5 T0=~T0;//每 10ns 一次
        initial
        begin
            T0 = 1'b0;
            IR = 16'b00000_000_001_00000;//测试 ADD R0+R1
        end
    Decode testADD(T0,IR,valA,valB,X,Addr);
endmodule
```

13. Execute_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/08 16:14:40
// Design Name:
// Module Name: Execute_tb_add
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

/**
测试执行模块，
输入 IR 00000_000_001_00000，valA，valB，X，Addr
译码得到输出 ALUOUT
成功
```

```
*/
module Execute_tb_add();
    reg T1;
    reg [7:0] valA;
    reg [7:0] valB;
    reg [7:0] X;
    reg [15:0] Addr;
    reg [15:0] IR;
    wire [7:0] ALUOUT;
    always #5 T1=~T1;//每 10ns 一次
    initial
        begin
            T1 = 1'b0;
            valA = 8'b0000_0001;
            valB = 8'b0000_1110;
            X    = 8'b0000_1111;
            Addr = 16'b0000_0000_0000_0001;
            IR = 16'b00000_000_001_00000;//测试 ADD R0+R1 （valA+valB）
        end
    Execute testADD(T1,valA,valB,X,Addr,IR,ALUOUT);
endmodule
```