

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

[一、 指令格式设计； 3](#_Toc863)

[二、 微操作的定义； 4](#_Toc20101)

[（一） 取指阶段 4](#_Toc3878)

[（二） 执行阶段 4](#_Toc31480)

[说明： 5](#_Toc12951)

[三、 节拍的划分； 6](#_Toc19942)

[（一） 取指阶段 8](#_Toc23328)

[1． 取指周期T0 8](#_Toc25839)

[2． 取指周期T1 8](#_Toc1599)

[3． 取指周期T2 9](#_Toc11394)

[4． 取指周期T3 9](#_Toc2000)

[（二） 执行阶段 9](#_Toc1430)

[1． 译码周期T0 9](#_Toc12136)

[2． 运算周期T1 9](#_Toc4599)

[3． 访存周期T2 10](#_Toc11754)

[4． 写回周期T3 11](#_Toc9247)

[四、 处理器结构设计框图及功能描述； 12](#_Toc25733)

[（一） 处理器结构设计框图 12](#_Toc30390)

[（二） 处理器模块划分 13](#_Toc20649)

[（三） 各模块说明及功能描述 13](#_Toc12588)

[1． Clock模块 13](#_Toc9411)

[2． CPU内外接口管理模块 14](#_Toc10762)

[3． Fetch模块 15](#_Toc23488)

[4． Decode模块 16](#_Toc7407)

[5． Execute模块 17](#_Toc17821)

[6． Access模块 18](#_Toc31062)

[7． WriteBack模块 19](#_Toc15764)

[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； 20](#_Toc22671)

[（一） 操作时间表 20](#_Toc32426)

[（二） 每个控制信号逻辑图 22](#_Toc23710)

[1． FE----取指周期 22](#_Toc29096)

[2． EX----译码周期 25](#_Toc1742)

[3． EX----执行周期 27](#_Toc26554)

[4． EX----访存周期 29](#_Toc19528)

[5． EX----写回阶段 31](#_Toc3928)

[六、 加分项2：用Verilog实现该CPU，并仿真验证其功能。 33](#_Toc8170)

[（一） 设计文件，模块介绍 33](#_Toc7458)

[1． CPU(CPU.v) 33](#_Toc9252)

[2． CPU接口管理模块（CPU\_export） 33](#_Toc12491)

[3． 取指阶段（取指模块Fetch）： 34](#_Toc3264)

[4． 执行阶段(Decode模块) 34](#_Toc32340)

[5． 执行阶段(Execute模块) 34](#_Toc14251)

[6． 执行阶段(Access模块) 34](#_Toc17955)

[7． 执行阶段(WriteBack模块） 34](#_Toc32055)

[（二） 仿真文件介绍，分析 35](#_Toc15027)

[1． 对每个模块进行单独的仿真测试 35](#_Toc31620)

[2． 整体测试中，十条指令样例给定的数据环境 39](#_Toc10486)

[（三） verilog源代码 55](#_Toc32220)

[1． CPU.v 55](#_Toc8357)

[2． CPU\_export.v 58](#_Toc26701)

[3． Clock.v 62](#_Toc1026)

[4． Fetch.v 64](#_Toc4298)

[5． Execute.v 65](#_Toc25321)

[6． Access.v 67](#_Toc31383)

[7． WriteBack.v 70](#_Toc25372)

[8． CPU\_export\_read.v 72](#_Toc19917)

[9． CPU\_export\_write.v 73](#_Toc18475)

[10． Clock\_tb.v 75](#_Toc29746)

[11． Fetch\_tb.v 76](#_Toc32618)

[12． Decode\_tb.v 77](#_Toc19911)

[13． Execute\_tb.v 79](#_Toc30933)

**加分项1：IO接口，加分项2：verilog设计并仿真 均已完成**

# 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) |

# 微操作的定义；

## 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 |

## 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] → PC | [Zero(Reg(Ad1(IR))) \* Reg(R7)//Ad2(IR) + Nzero(Reg(Ad1(IR))] +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

# 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 Ri, Rj  加法指令  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)) |
| 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] → 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(Addr)->Rtemp   1. >R   1 -> Mem | Rtemp -> Reg(Ad1(IR)) |
| 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 |

## 取指阶段

### 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 |

### 取指周期T1

|  |  |
| --- | --- |
| 指令 | 取指周期T1 |
| 所有指令 | PC+1 ->PC |

### 取指周期T2

|  |  |
| --- | --- |
| 指令 | 取指周期T2 |
| 所有指令 | 空白 |

### 取指周期T3

|  |  |
| --- | --- |
| 指令 | 取指周期T3 |
| 所有指令 | 空白 |

## 执行阶段

### 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 |

### 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 |

### 访存周期T2

|  |  |
| --- | --- |
| 指令 | **访存周期T2** |
| ADD Ri, Rj  加法指令  Ri + Rj → Ri | 空白 |
| 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 |

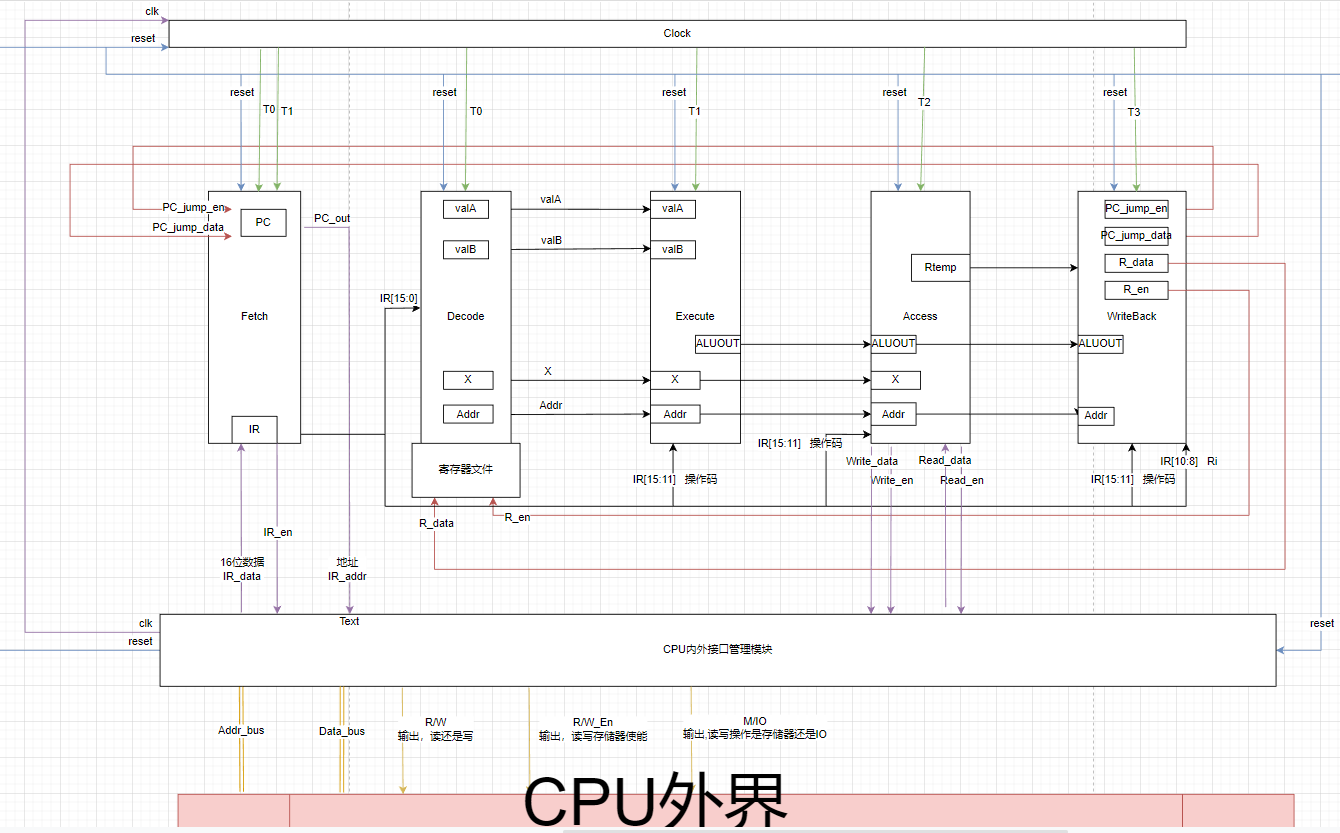
### 写回周期T3

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

# 处理器结构设计框图及功能描述；

## 处理器结构设计框图

**原图具体打印出来了附在后面**



## 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 |

## 各模块说明及功能描述

### 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 |
|

### 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 |

### 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 |
| 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 |

### 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 |

### 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 |

### 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 |

### 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 |

# 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；

## 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 |  |  |  |  |  |  |  |  |  |
| T1 | valA-valB -> ALUOUT |  | 1 |  |  |  |  |  |  |  |  |
| 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 |
| WriteBack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 |  |  |

## 每个控制信号逻辑图

### FE----取指周期

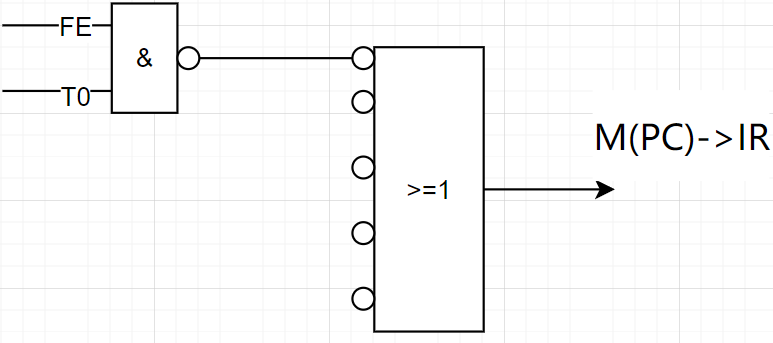
**控制信号:**

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:**

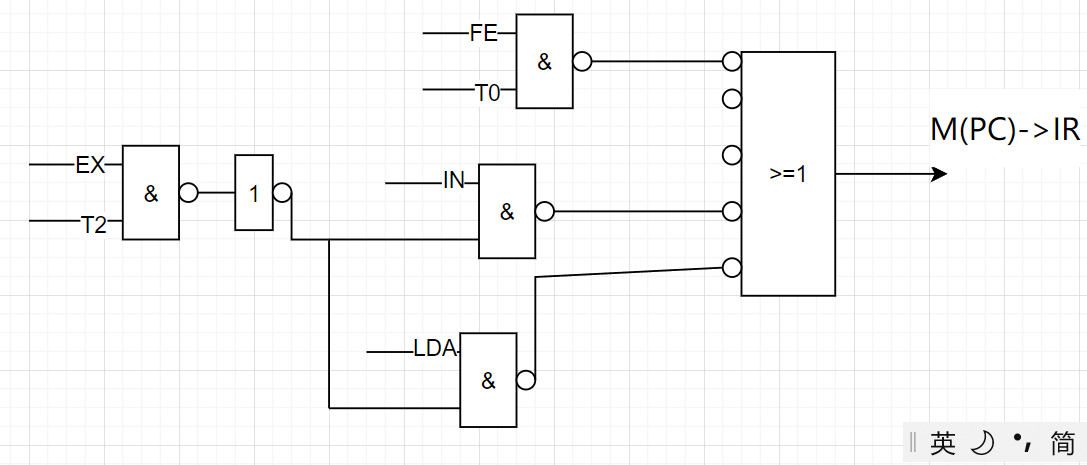
**控制信号:**

1 ->W

**微操作命令的最简表达式:**

=EX\*T2(OUT+STA)

**控制信号逻辑图:**



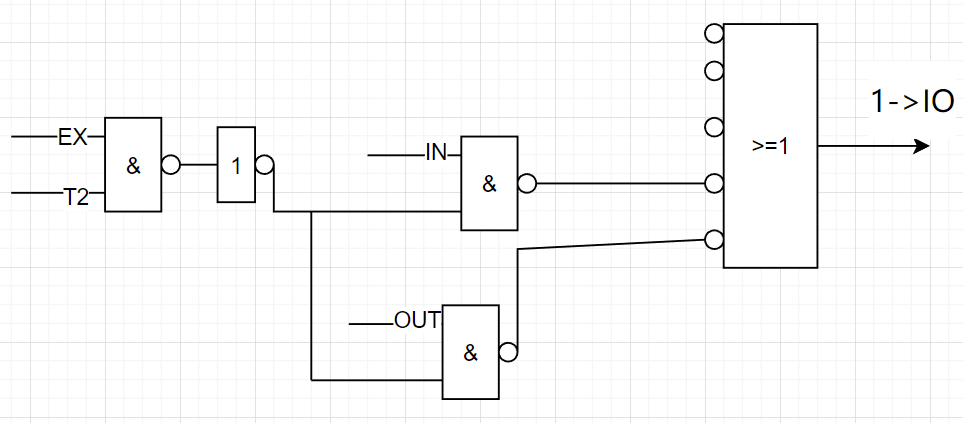
**控制信号:**

1 ->IO

**微操作命令的最简表达式:**

=EX\*T2(IN+OUT)

**控制信号逻辑图:**



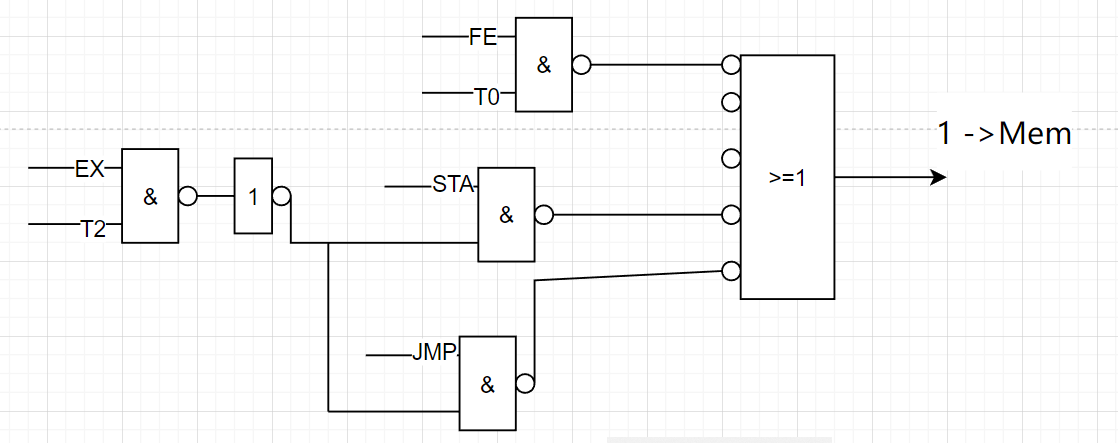
**控制信号:**

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:**



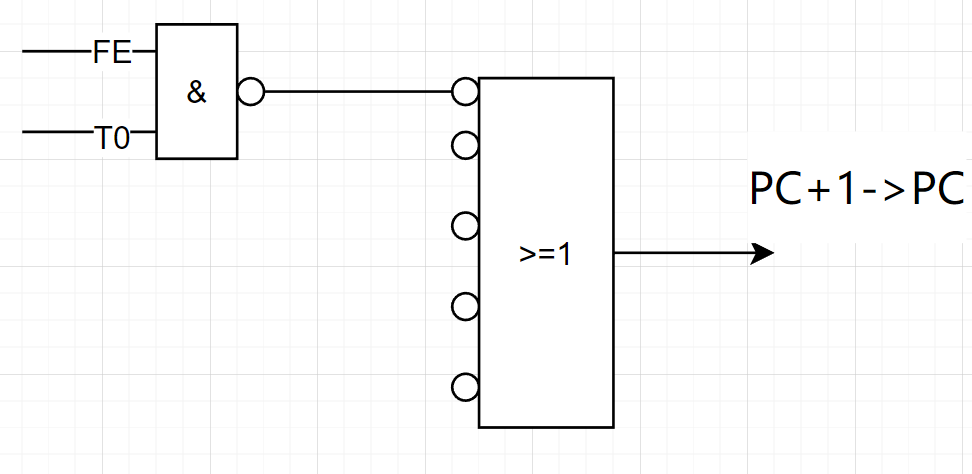
**控制信号:**

PC+1 ->PC

**微操作命令的最简表达式:**

=FE\*T0

**控制信号逻辑图:**



### EX----译码周期

**控制信号:**

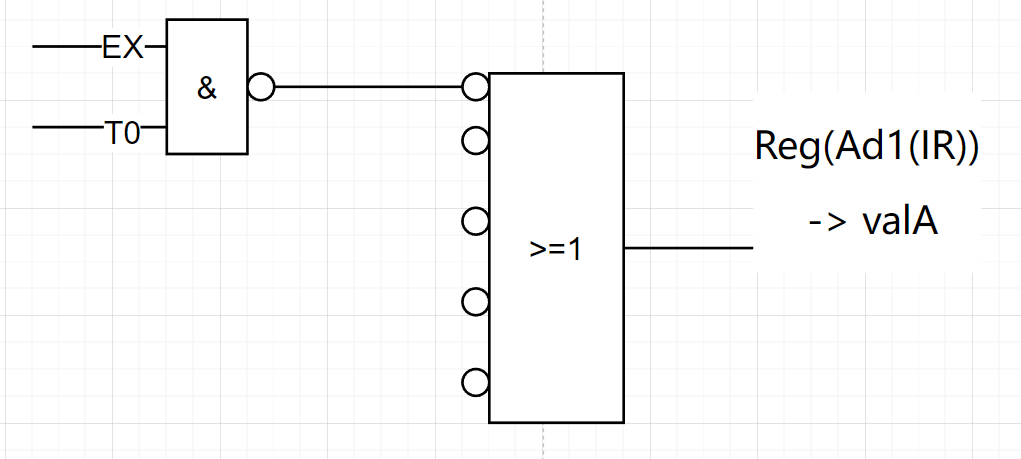
Reg(Ad1(IR))

-> 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:**

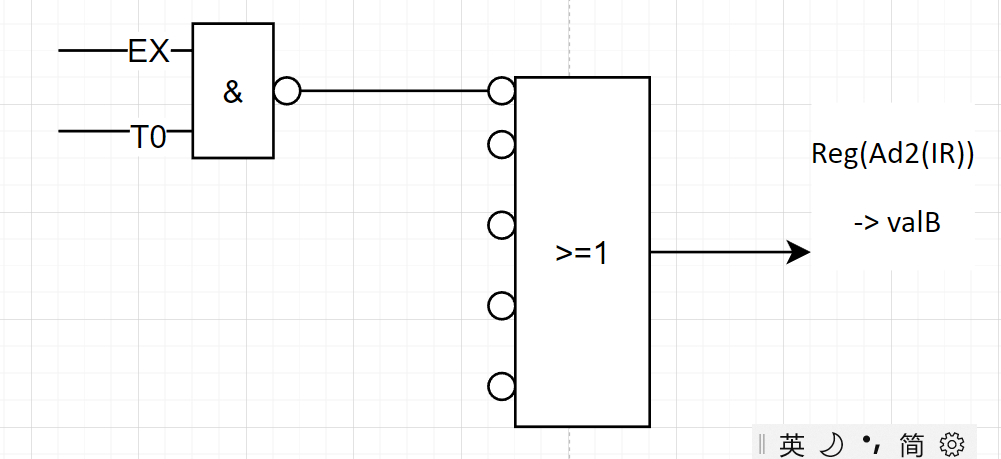
Reg(Ad2(IR))

-> 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:**

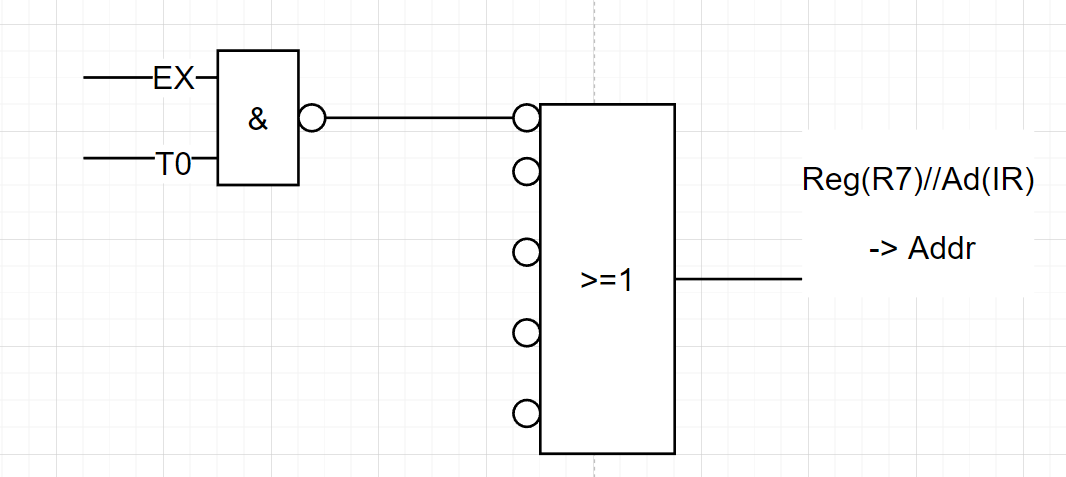
Reg(R7)//Ad(IR)

-> Addr

**微操作命令的最简表达式:**

=EX\*T0

**控制信号逻辑图:**



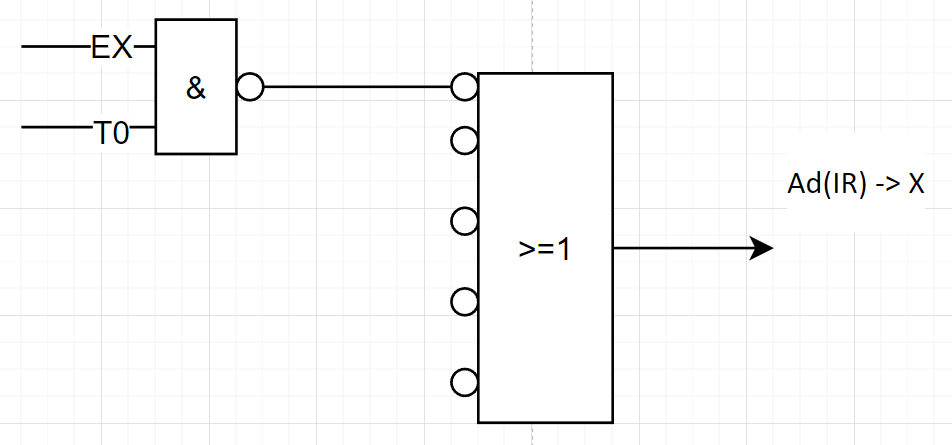
**控制信号:**

Ad(IR) -> X

**微操作命令的最简表达式:**

=EX\*T0

**控制信号逻辑图:**



### EX----执行周期

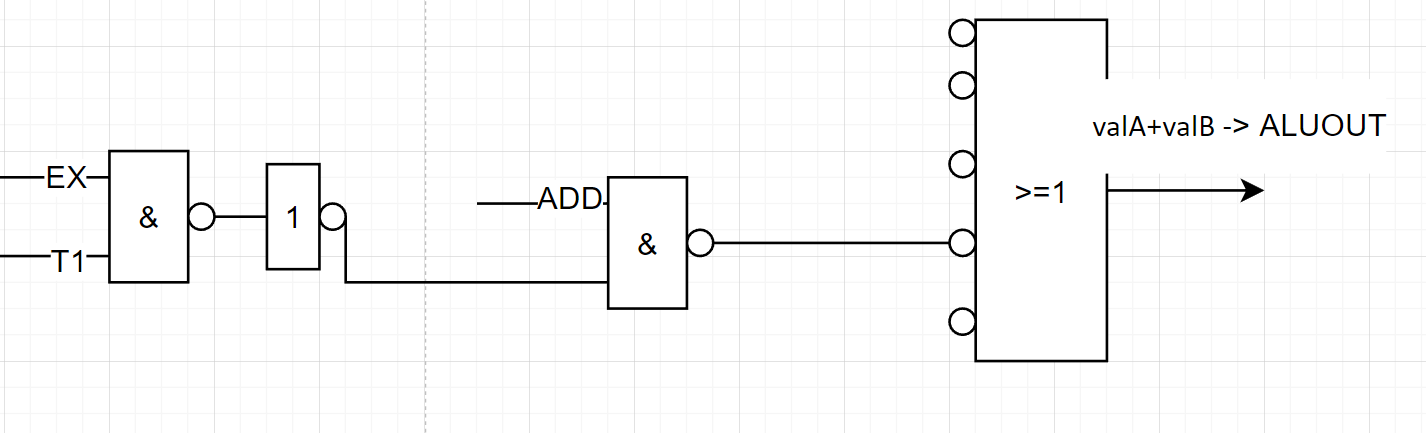
**控制信号:**

valA+valB -> ALUOUT

**微操作命令的最简表达式:**

=EX\*T1\*(ADD)

**控制信号逻辑图:**



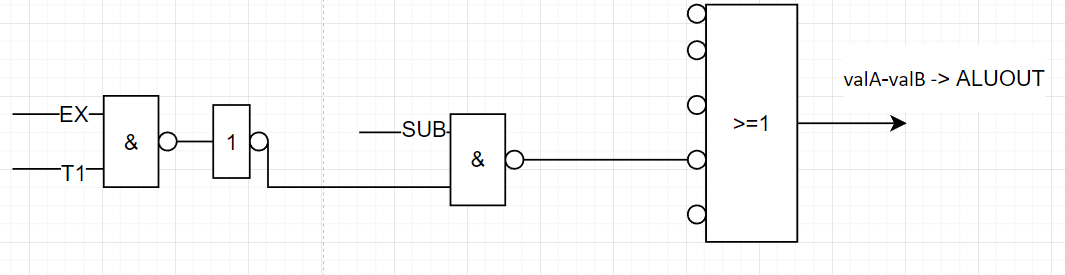
**控制信号:**

valA-valB -> ALUOUT

**微操作命令的最简表达式:**

=EX\*T1\*(SUB)

**控制信号逻辑图:**



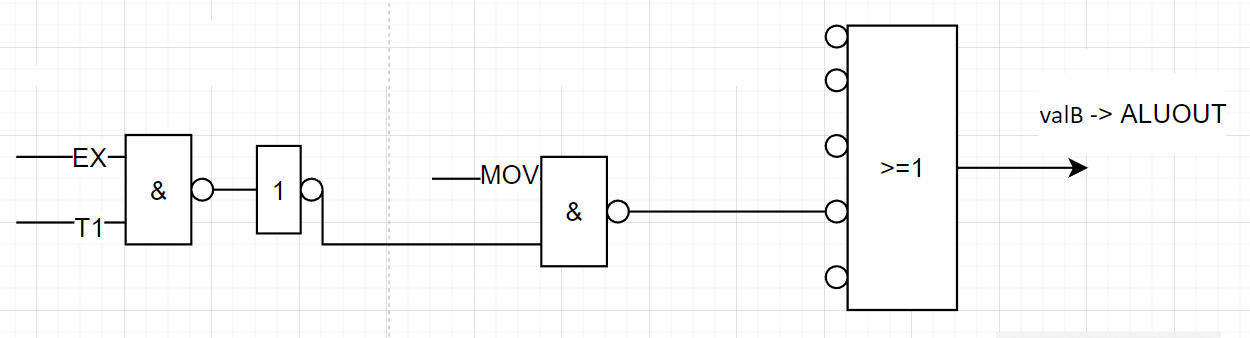
**控制信号:**

valB -> ALUOUT

**微操作命令的最简表达式:**

=EX\*T1\*(MOV)

**控制信号逻辑图:**



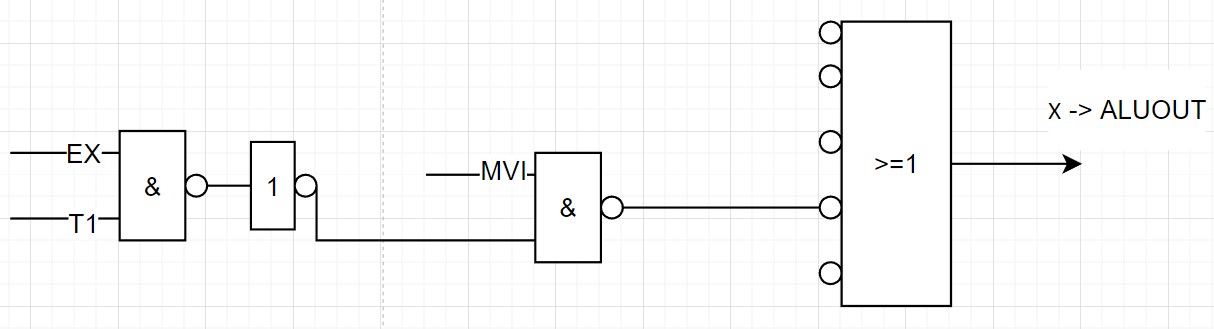
**控制信号:**

X -> ALUOUT

**微操作命令的最简表达式:**

=EX\*T1\*(MVI)

**控制信号逻辑图:**



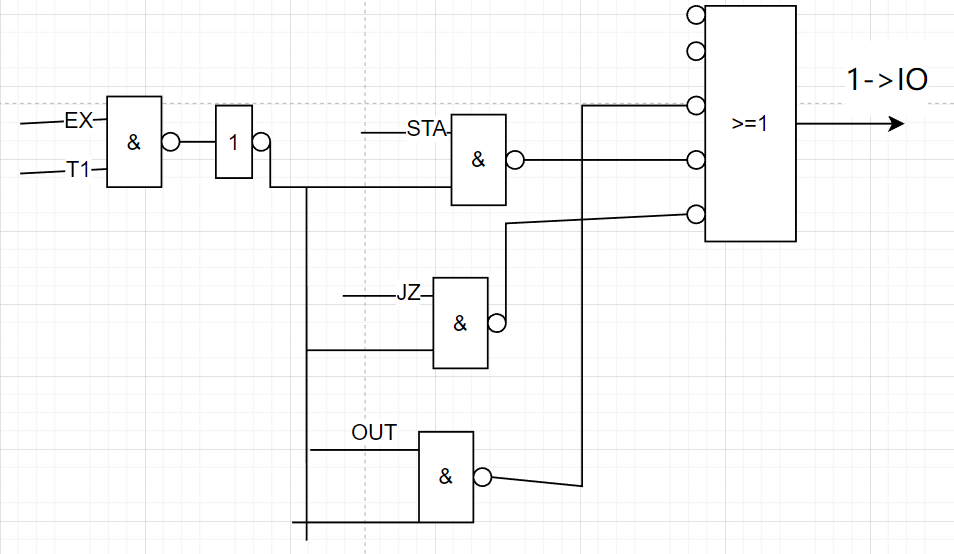
**控制信号:**

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:**



### EX----访存周期

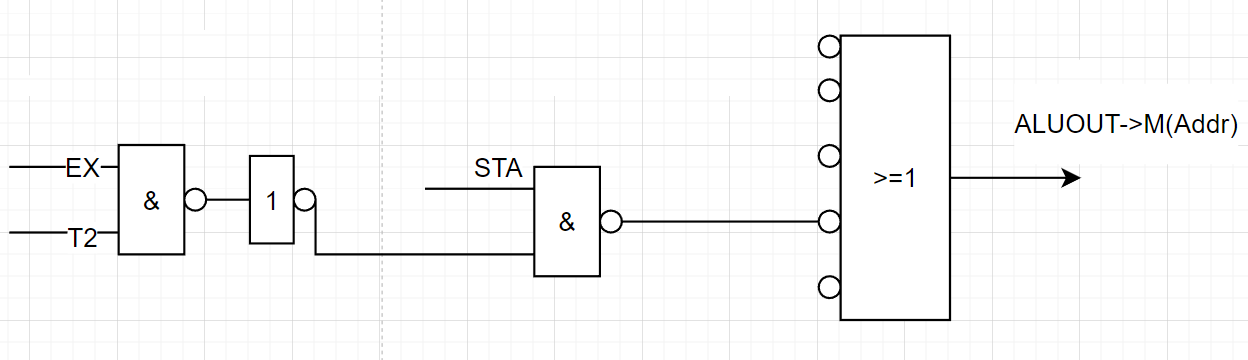
**控制信号:**

ALUOUT->M(Addr)

**微操作命令的最简表达式:**

=EX\*T2\*（STA)

**控制信号逻辑图:**



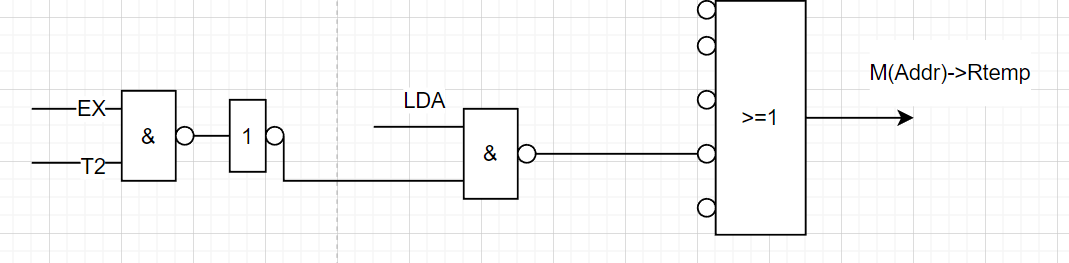
**控制信号:**

M(Addr)->Rtemp

**微操作命令的最简表达式:**

=EX\*T2\*（LDA)

**控制信号逻辑图:**



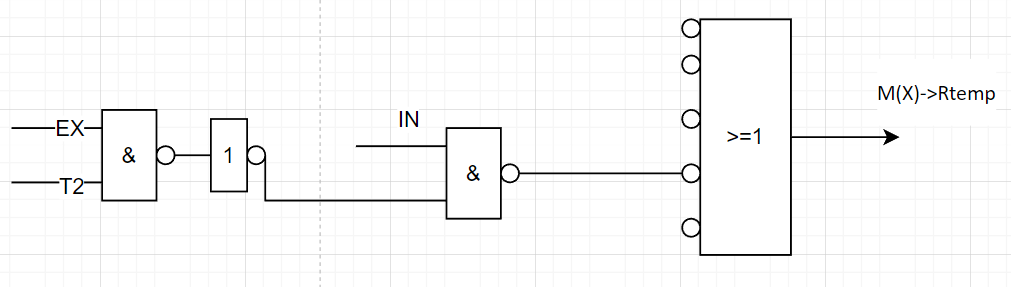
**控制信号:**

M(X)->Rtemp

**微操作命令的最简表达式:**

=EX\*T2\*（IN)

**控制信号逻辑图:**



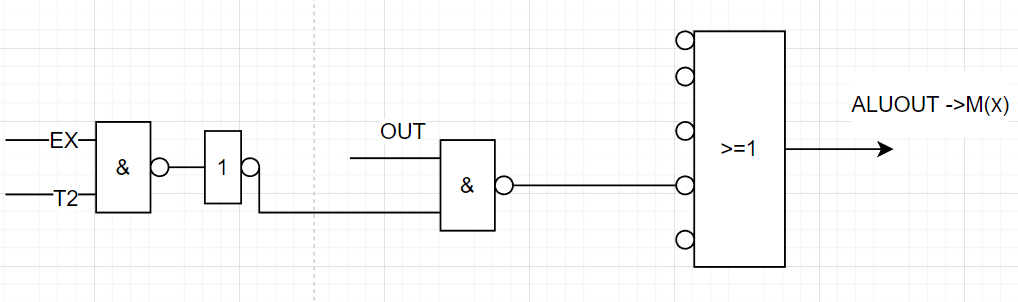
**控制信号:**

ALUOUT ->M(X)

**微操作命令的最简表达式:**

=EX\*T2\*（OUT)

**控制信号逻辑图:**



### EX----写回阶段

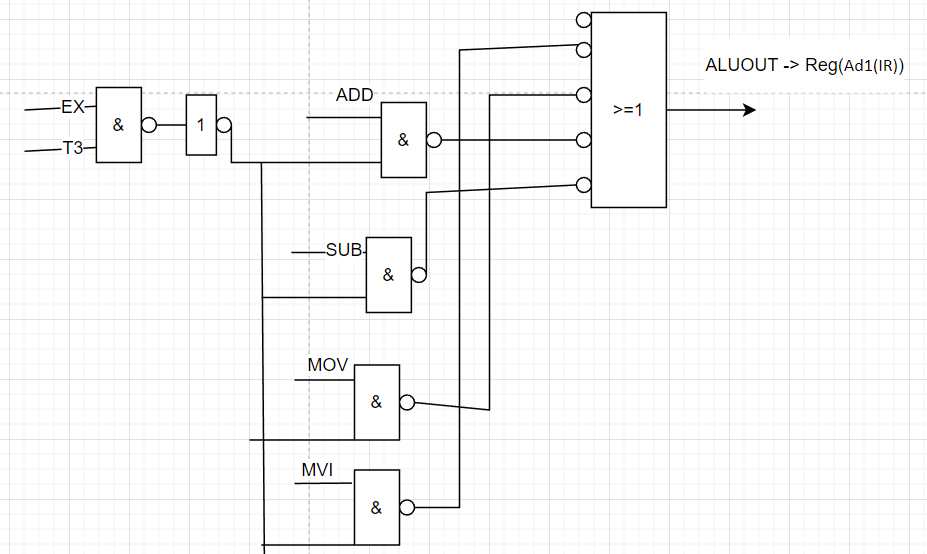
**控制信号:**

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:**



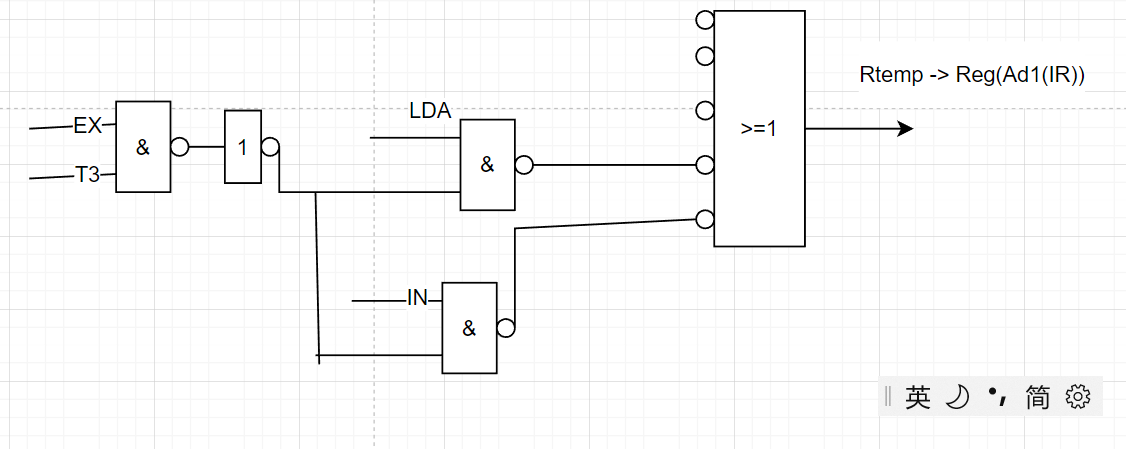
**控制信号:**

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

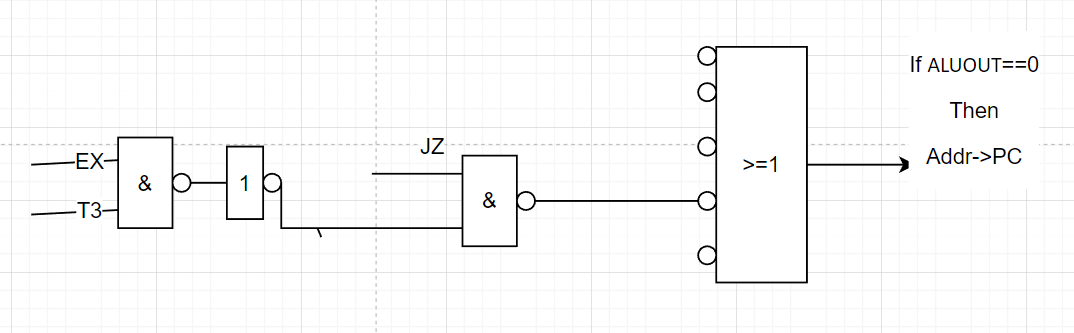
Then

Addr->PC

**微操作命令的最简表达式:**

=EX\*T3 \*(JZ)

**控制信号逻辑图:**



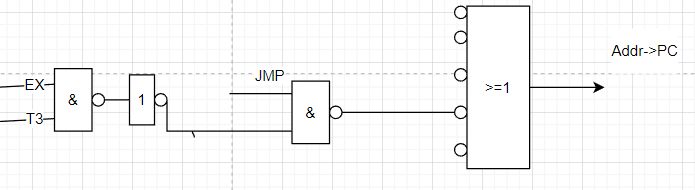
**控制信号:**

Addr->PC

**微操作命令的最简表达式:**

=EX\*T3 \*(JMP)

**控制信号逻辑图:**



# 加分项2：用Verilog实现该CPU，并仿真验证其功能。

## 设计文件，模块介绍

### 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

### 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

### 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

### 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

### 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

### 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。

### 执行阶段(WriteBack模块）

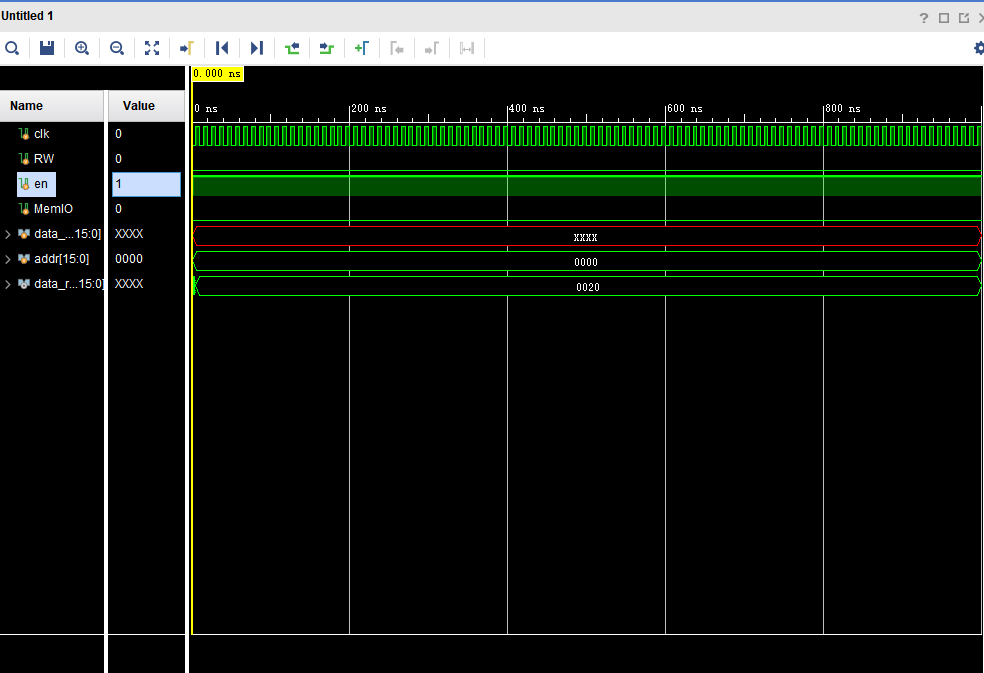
T3周期：根据输入的IR,ALUOUT,Rtemp,Addr,T3，根据不同的指令将需要更新的数值更新对应的寄存器，则会输出R\_select写入的寄存器的编号,R\_data写入的寄存器的值,R\_en写入的使能端

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

## 仿真文件介绍，分析

### 对每个模块进行单独的仿真测试

**CPU\_export\_read**



测试CPU接口模块

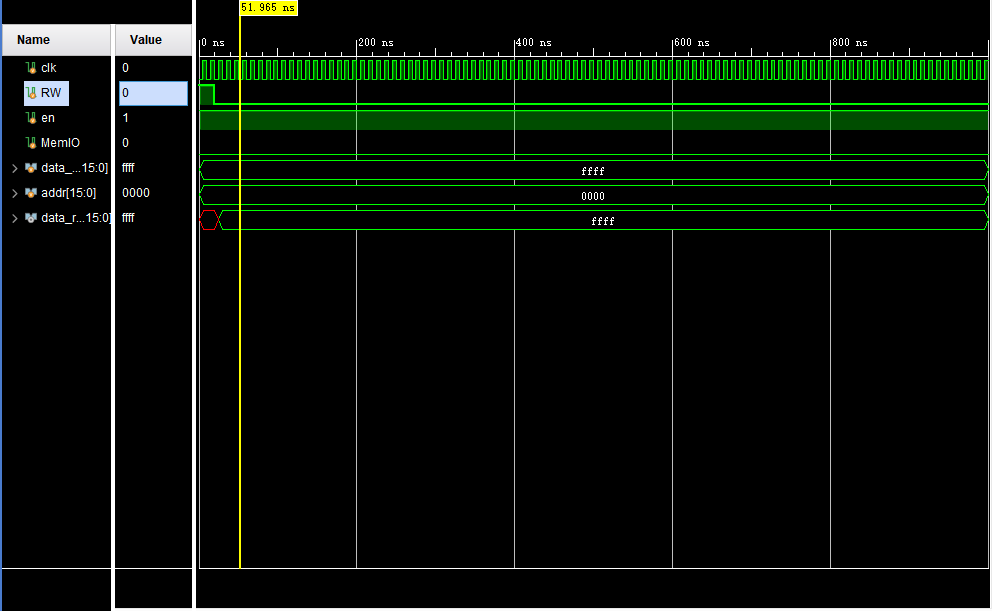
输入addr 0000\_0000\_0000\_0000

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

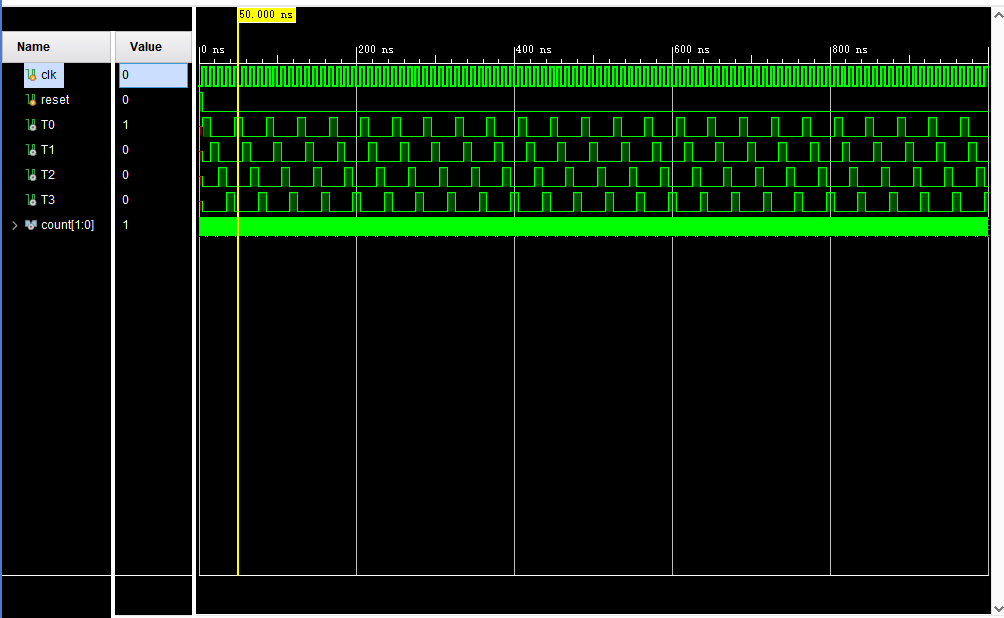
向该地址写入data\_write 11111\_111\_111\_11111

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**

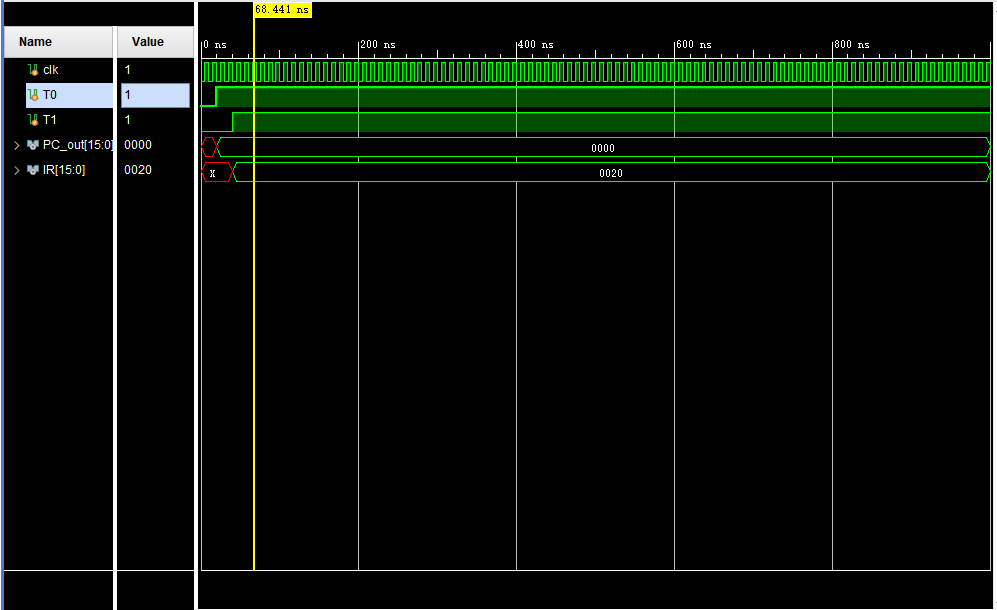


测试Clock模块

让其产生周期T0~T3

**成功**

**Fetch\_tb**



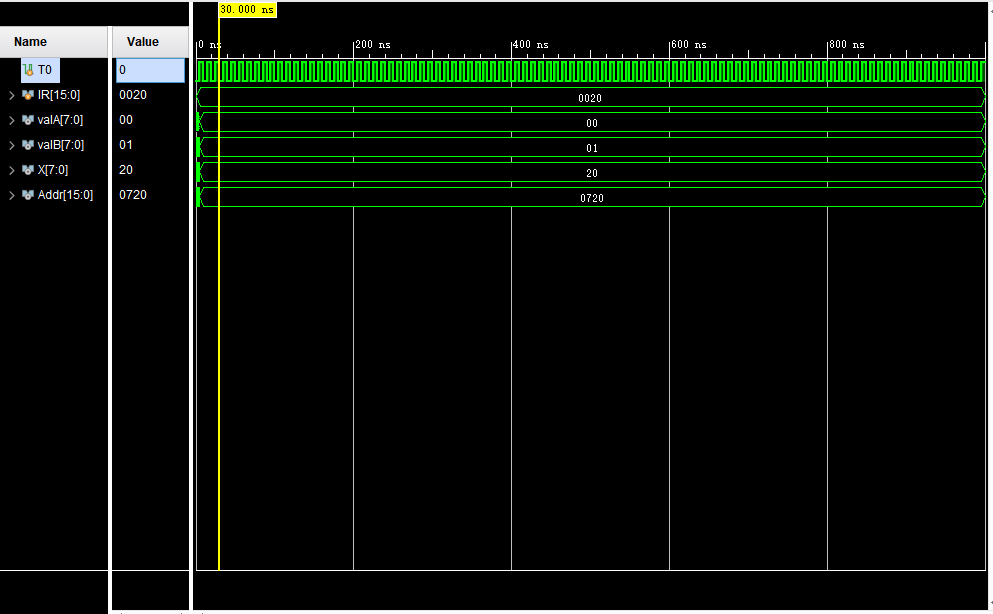
测试取指模块

PC为默认16'b0000\_0000\_0000\_0000

取出IR指令00000\_000\_001\_00000

**成功**

**Decode\_tb**



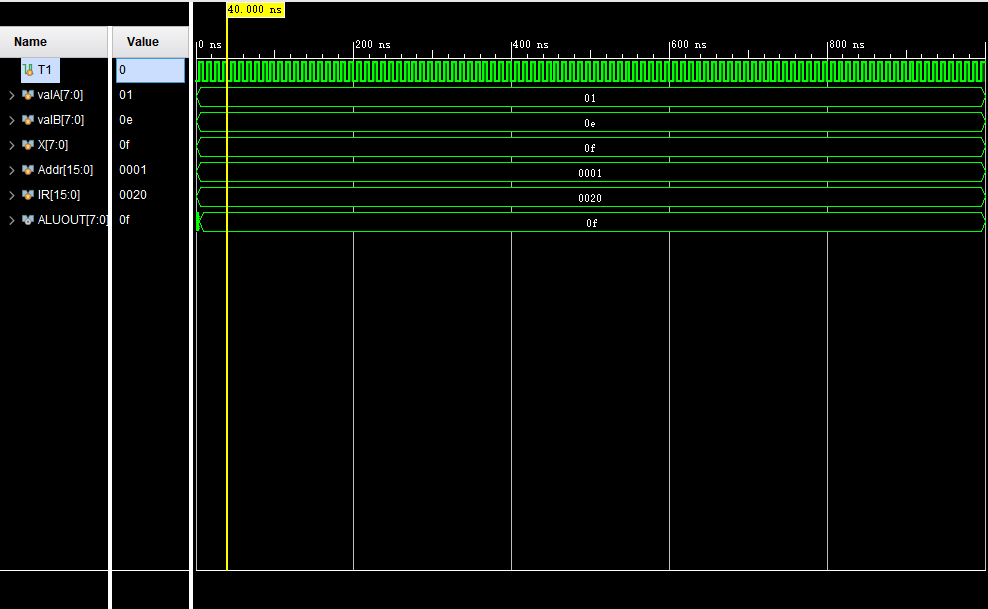
测试译码模块，

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**

### 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; |

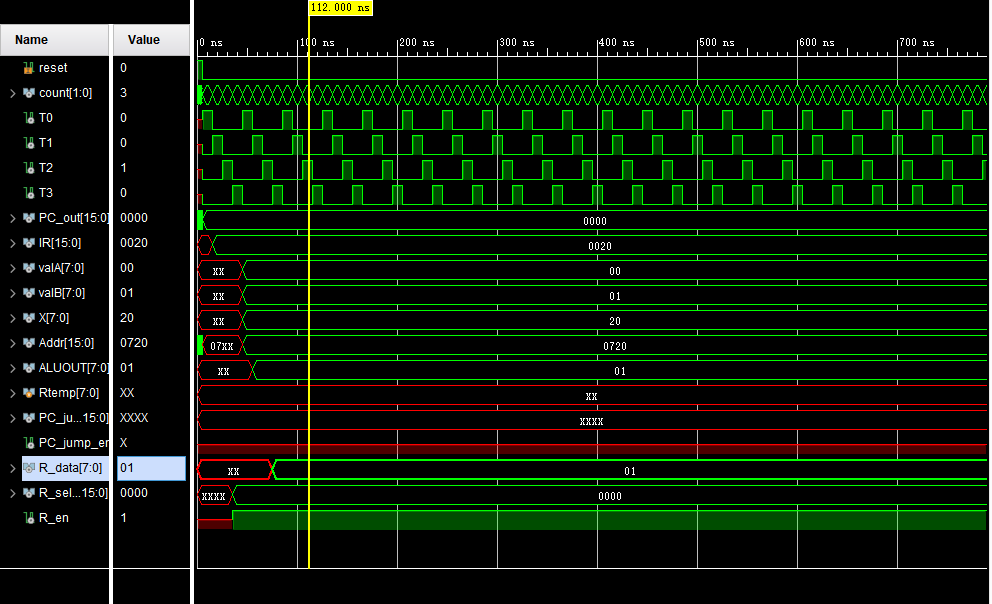
**每个模块单独测试本模块功能：**

**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：

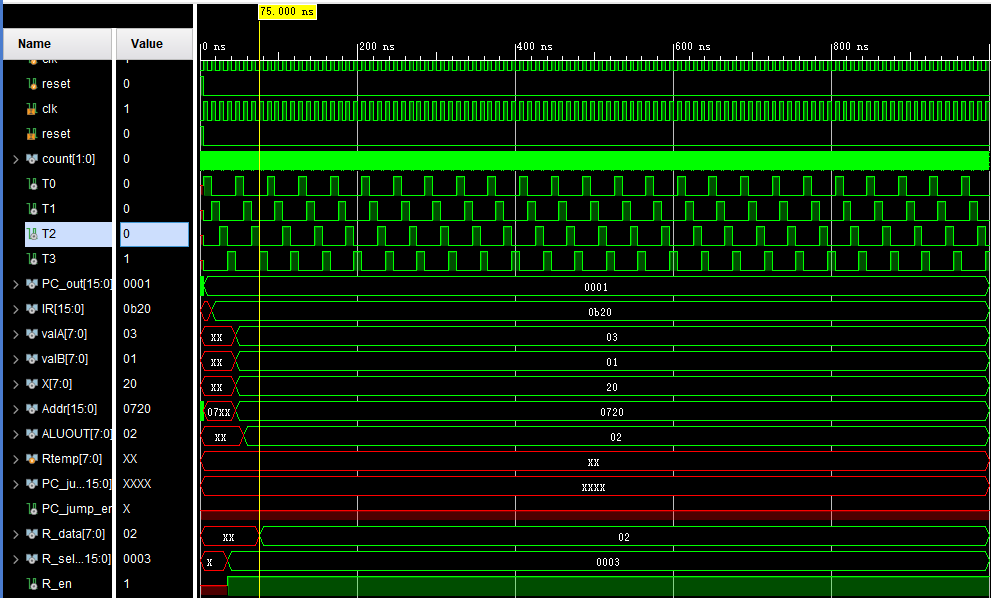
发出信号使得将ALUOUT的值可以写回寄存器文件修改R0:

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，

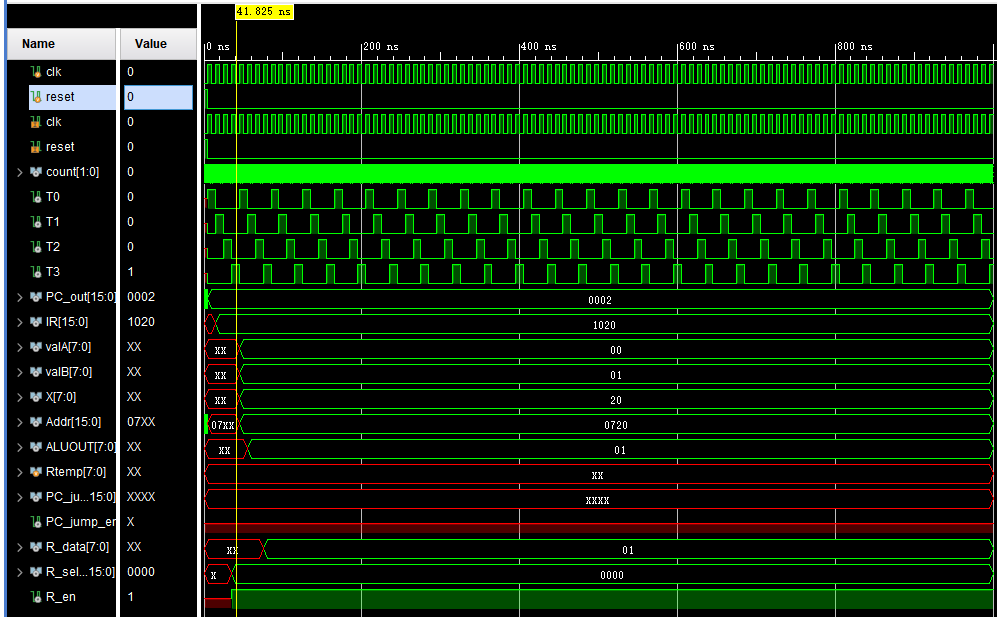
可以写回寄存器文件修改R3:

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：

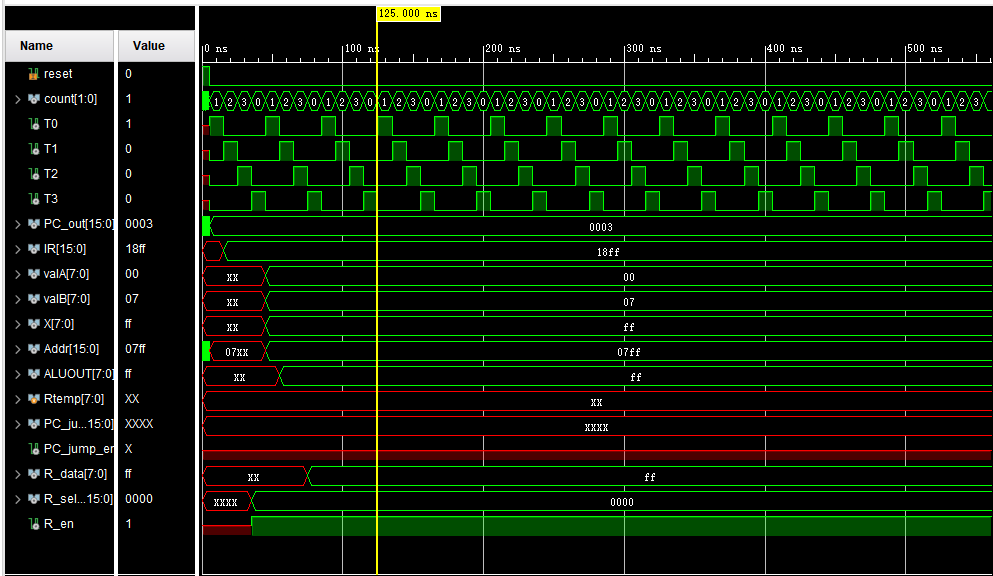
发出信号可以写回寄存器文件修改R0

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：

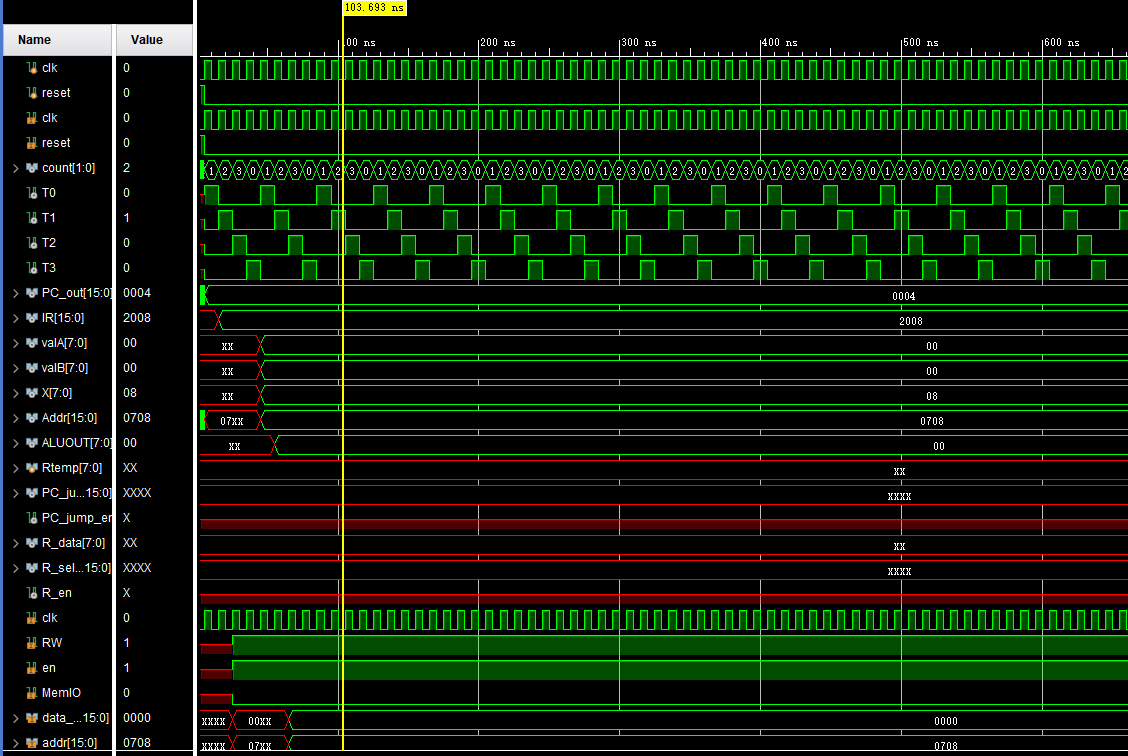
发出信号可以写回寄存器文件修改R0

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

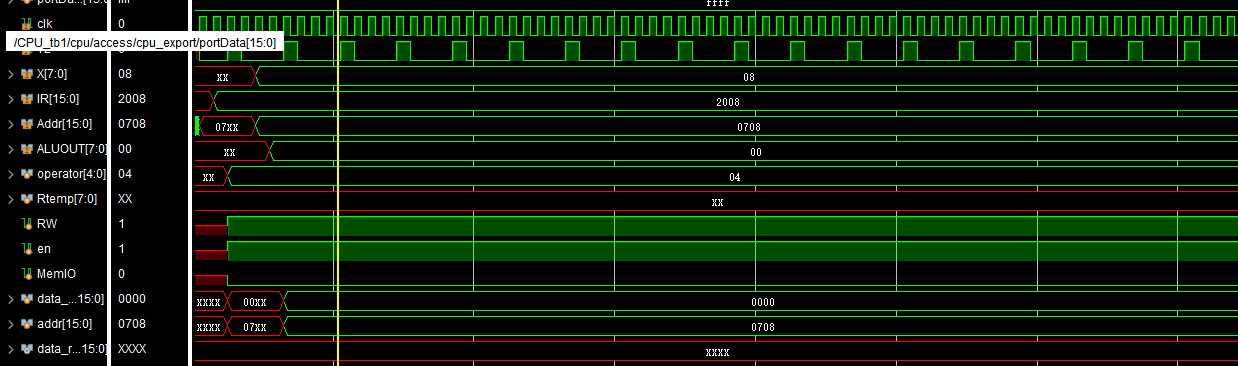
Addr=[R7//X] = :0708h

**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;

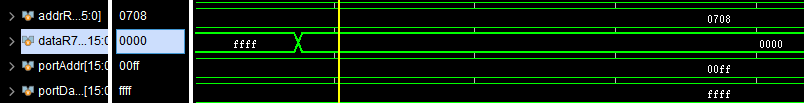
addr = Addr = 0708h;

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

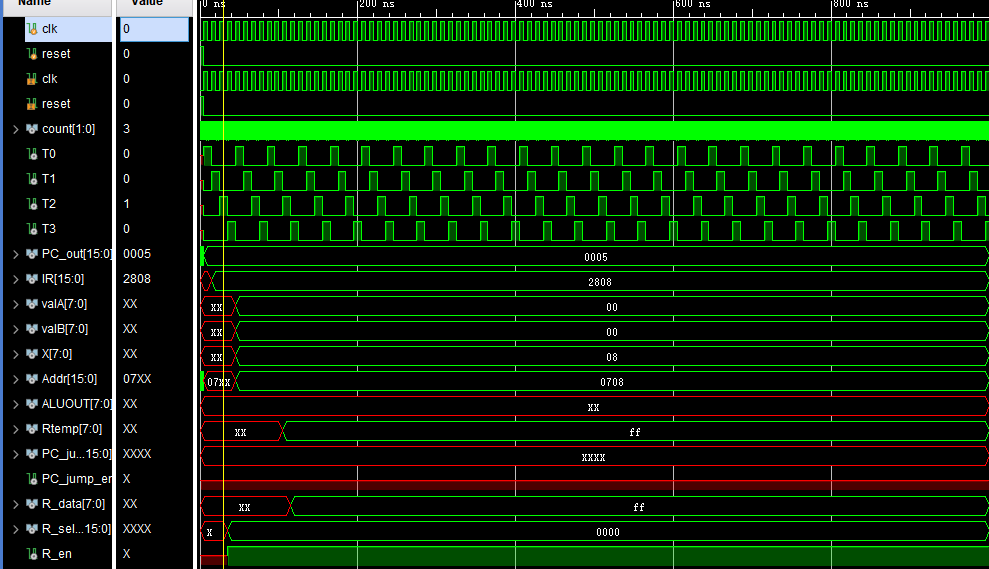
数据修改为了0000

**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

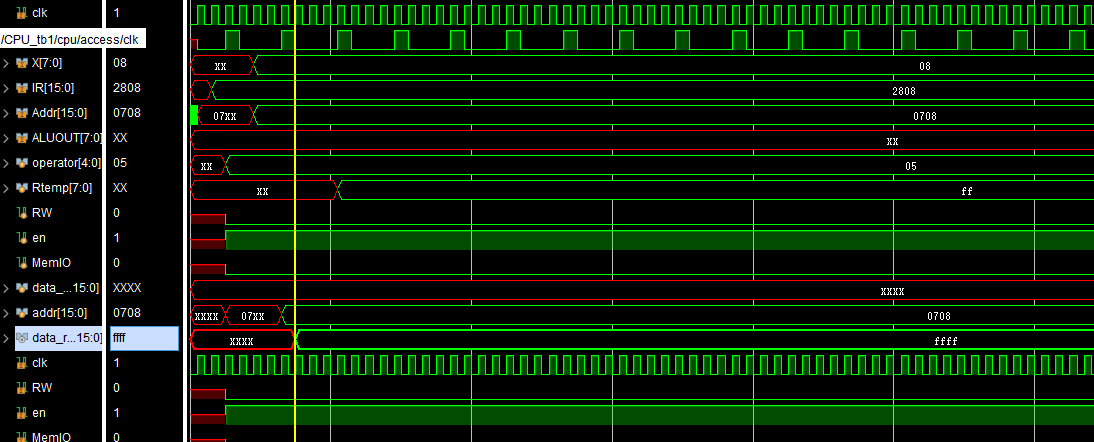
Addr=[R7//X] = :0708h

**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;

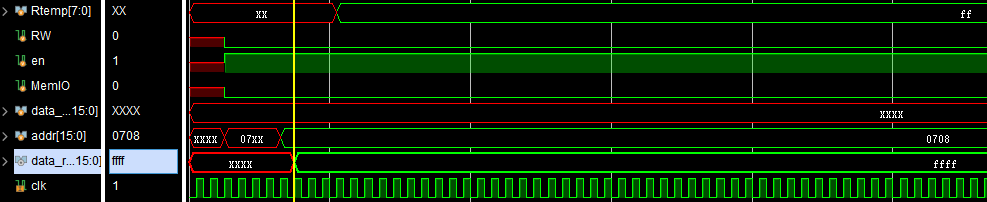
en=1;

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

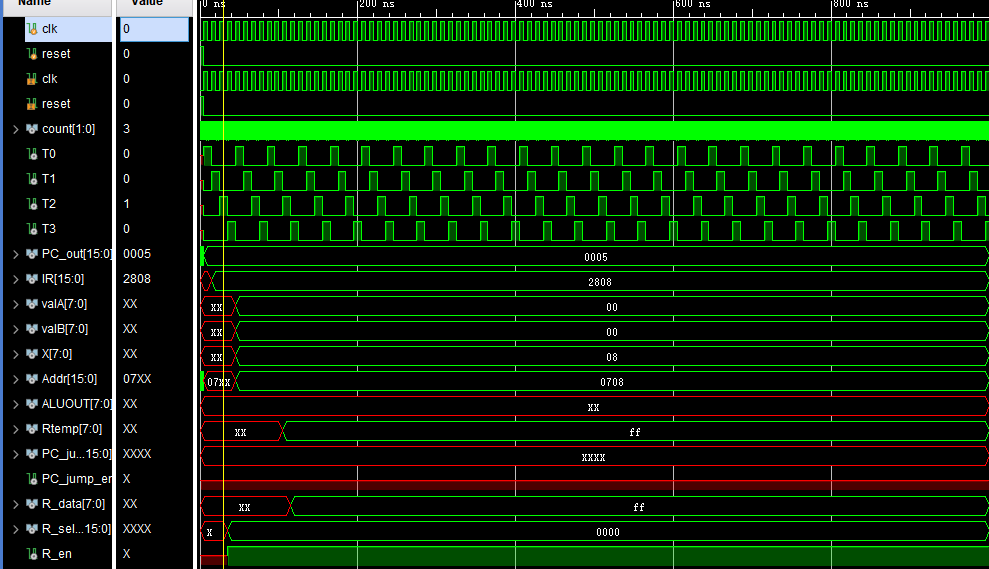


发现在T2时刻之后

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

1. 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

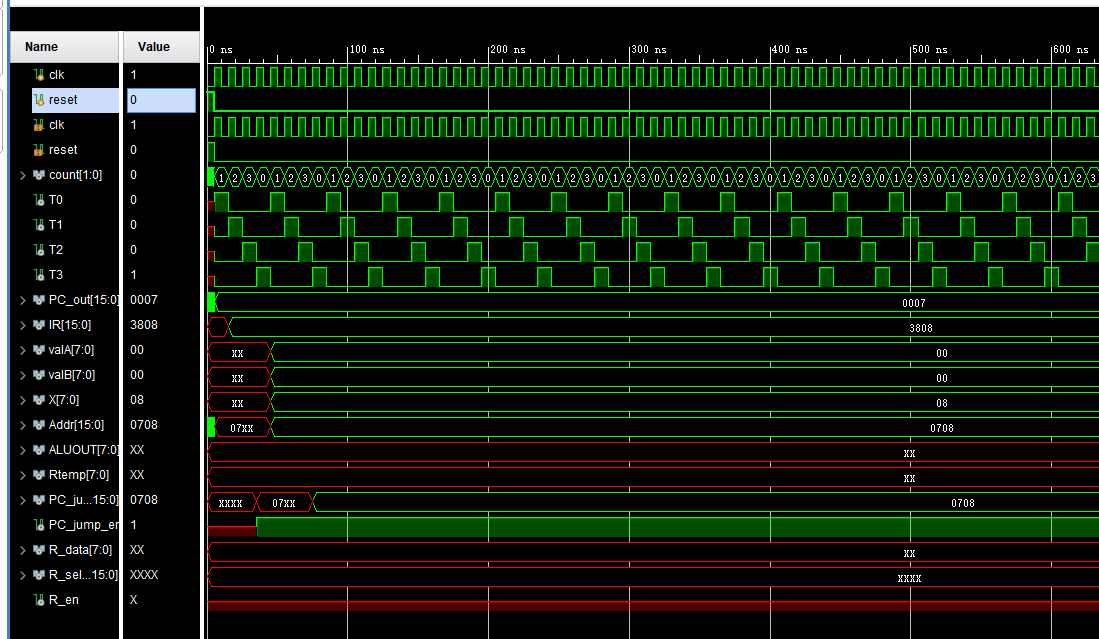
发出信号需要执行跳转指令

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：

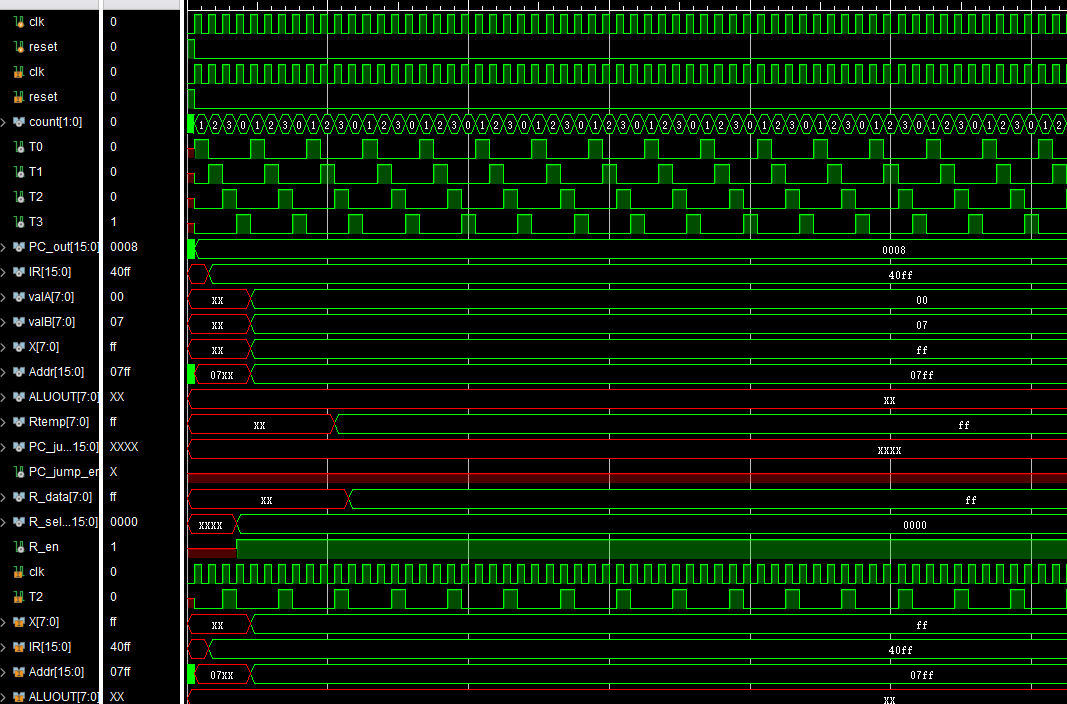
发出信号需要执行跳转指令

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

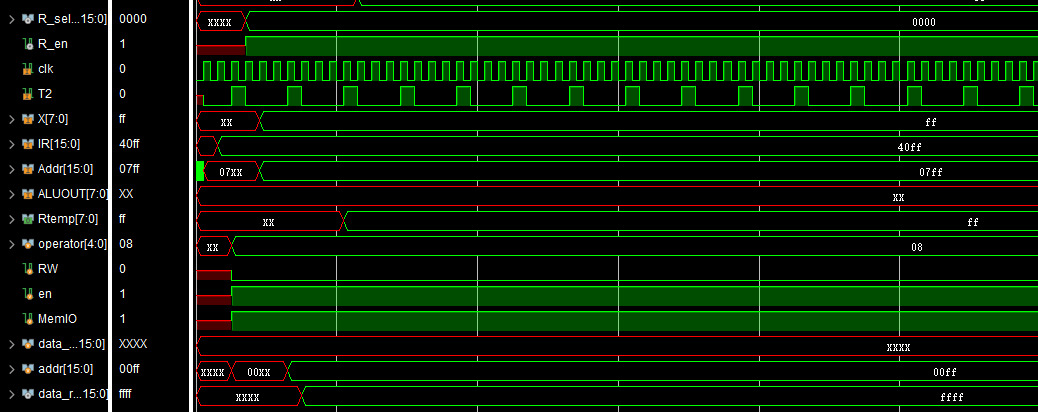
**执行阶段(Execute模块)**

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

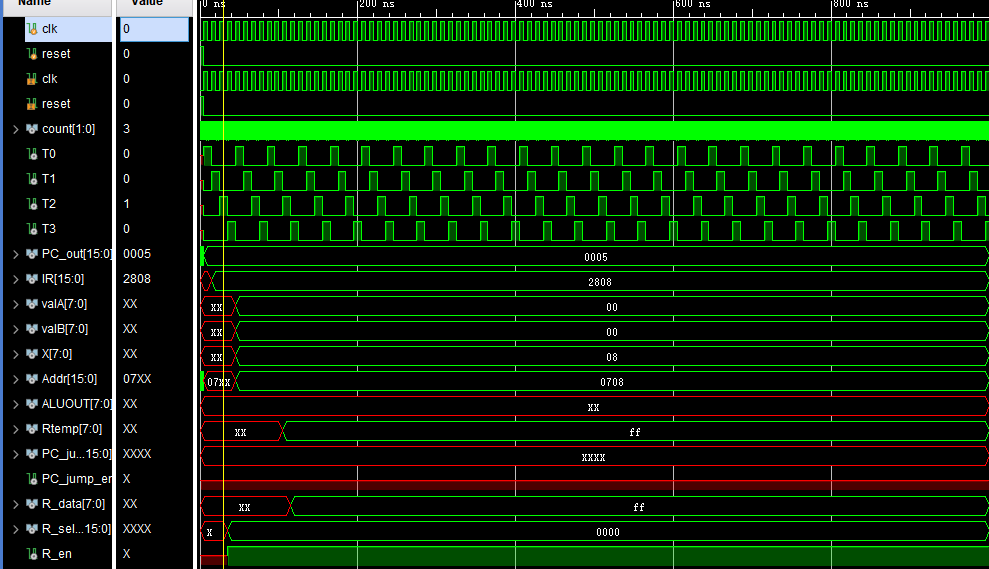


发现在T2时刻之后

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

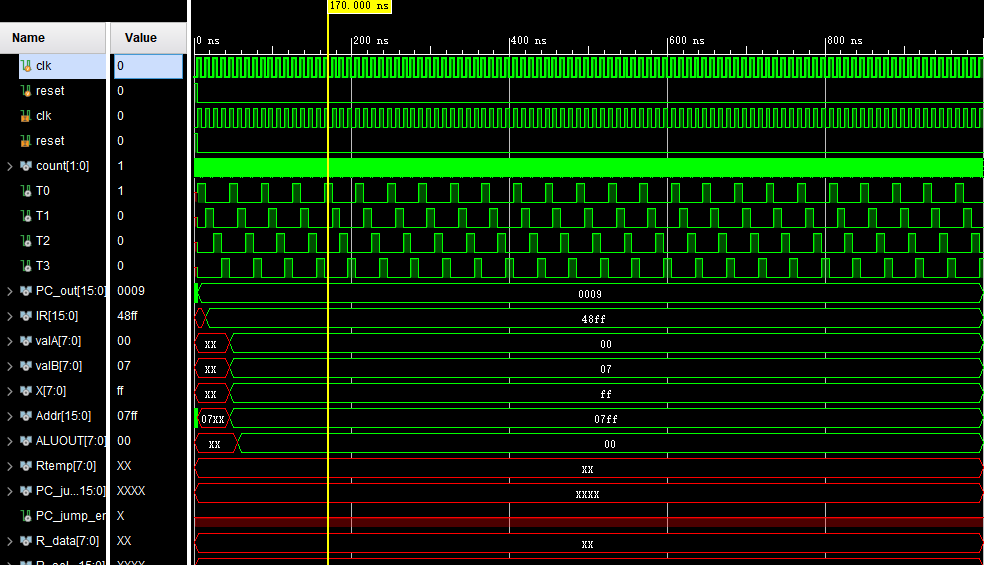
然后发出信号Rdata = ffh

1. 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

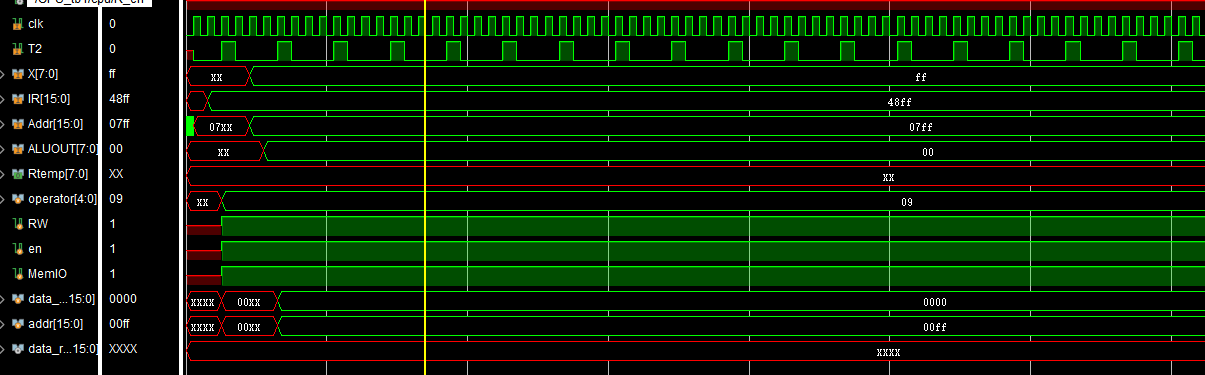
Addr=[R7//X] = :07ffh

**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)

1. >W

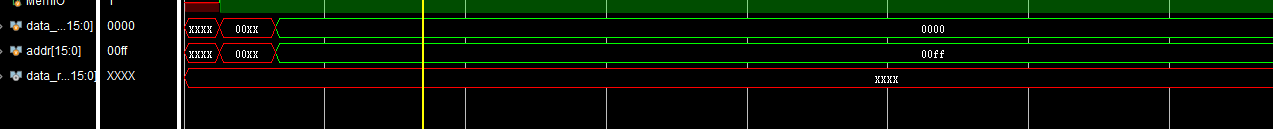
1 -> IO

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

## verilog源代码

### CPU.v

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2022/11/07 16:51:42*
7. *// Design Name:*
8. *// Module Name: CPU*
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. module CPU(
22. input wire clk,
23. input wire reset
24. );
25. wire [1:0] count;
26. wire T0;
27. wire T1;
28. wire T2;
29. wire T3;
30. wire [15:0] PC\_out;*//往外输出的PC*
31. wire [15:0] IR;
32. wire [7:0] valA;
33. wire [7:0] valB;
34. wire [7:0] X;
35. wire [15:0] Addr;
36. wire [7:0] ALUOUT;
37. wire [7:0] Rtemp;
38. wire [15:0] PC\_jump\_data;
39. wire  PC\_jump\_en;
40. wire [7:0] R\_data;
41. wire [15:0] R\_select;
42. wire  R\_en;
43. Clock clock(
44. .clk(clk),
45. .reset(reset),
46. .count(count),
47. .T0(T0),
48. .T1(T1),
49. .T2(T2),
50. .T3(T3)
51. );
52. Fetch fetch(
53. .clk(clk),
54. .T0(T0),
55. .T1(T1),
56. .PC\_out(PC\_out),
57. .IR(IR)
58. );
59. Decode decode(
60. .T0(T0),
61. .IR(IR),
62. .valA(valA),
63. .valB(valB),
64. .X(X),
65. .Addr(Addr)
66. );
67. Execute execute(
68. .T1(T1),
69. .valA(valA),
70. .valB(valB),
71. .X(X),
72. .Addr(Addr),
73. .IR(IR),
74. .ALUOUT(ALUOUT)
75. );
76. Access access(
77. .clk(clk),
78. .T2(T2),
79. .X(X),
80. .IR(IR),
81. .Addr(Addr),
82. .ALUOUT(ALUOUT),
83. .Rtemp(Rtemp)
84. );
85. WriteBack writeBack(
86. .T3(T3),
87. .Rtemp(Rtemp),
88. .ALUOUT(ALUOUT),
89. .Addr(Addr),
90. .IR(IR),
91. .PC\_jump\_data(PC\_jump\_data),
92. .PC\_jump\_en(PC\_jump\_en),
93. .R\_data(R\_data),
94. .R\_select(R\_select),
95. .R\_en(R\_en)
96. );
97. endmodule

### CPU\_export.v

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2022/11/08 18:26:04
7. // Design Name:
8. // Module Name: CPU\_export
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. module CPU\_export(
22. input clk,
23. input wire RW,      //0->R,1->W
24. input wire en,      //en =1 代表可以
25. input wire MemIO,//这次是访问IO还是Mem,0->mem,1->IO
26. input wire [15:0] data\_write,
27. input wire [15:0] addr,
28. output reg [15:0] data\_read
29. );
30. //模拟存储器
31. reg [15:0] addr0;
32. reg [15:0] data0;
33. reg [15:0] addr1;
34. reg [15:0] data1;
35. reg [15:0] addr2;
36. reg [15:0] data2;
37. reg [15:0] addr3;
38. reg [15:0] data3;
39. reg [15:0] addr4;
40. reg [15:0] data4;
41. reg [15:0] addr5;
42. reg [15:0] data5;
43. reg [15:0] addr6;
44. reg [15:0] data6;
45. reg [15:0] addr7;
46. reg [15:0] data7;
47. reg [15:0] addr8;
48. reg [15:0] data8;
49. reg [15:0] addr9;
50. reg [15:0] data9;
51. reg [15:0] addrR7X;
52. reg [15:0] dataR7X;
53. //模拟IO接口
54. reg [15:0] portAddr;
55. reg [15:0] portData;
56. initial
57. begin
58. //主存中存的PC地址和指令
59. addr0 = 16'b0000\_0000\_0000\_0000;//地址
60. data0 = 16'b00000\_000\_001\_00000;//ADD R0+R1                       //PC指令数据
61. addr1 = 16'b0000\_0000\_0000\_0001;//地址
62. data1 = 16'b00001\_011\_001\_00000;//SUB R3-R1                       //PC指令数据
63. addr2 = 16'b0000\_0000\_0000\_0010;//地址
64. data2 = 16'b00010\_000\_001\_00000;//MOV (Rj)R1 -> R0(Ri)                       //PC指令数据
65. addr3 = 16'b0000\_0000\_0000\_0011;//地址
66. data3 = 16'b00011\_000\_111\_11111;//MVI X(1111\_1111) -> Ri(R0)                       //PC指令数据
67. addr4 = 16'b0000\_0000\_0000\_0100;//地址
68. data4 = 16'b00100\_000\_0000\_1000;//STA R0 -> [R7//X] (X=0000\_1000)                       //PC指令数据
69. addr5 = 16'b0000\_0000\_0000\_0101;//地址
70. data5 = 16'b00101\_000\_0000\_1000;//LDA [R7//X] -> Ri(R0) (X=0000\_1000)                      //PC指令数据
71. addr6 = 16'b0000\_0000\_0000\_0110;//地址      (R0确实为0)
72. data6 = 16'b00110\_000\_0000\_1000;//JZ if(Ri Ro == 0) then [R7//X]->PC             (X=0000\_1000)           //PC指令数据
73. addr7 = 16'b0000\_0000\_0000\_0111;//地址
74. data7 = 16'b00111\_000\_0000\_1000;//JMP [R7//X] -> PC  (X=0000\_1000)                       //PC指令数据
75. addr8 = 16'b0000\_0000\_0000\_1000;//地址
76. 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
77. addr9 = 16'b0000\_0000\_0000\_1001;//地址
78. 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
79. //[R7//X]会涉及到的地址
80. addrR7X = 16'b0000\_0111\_0000\_1000;
81. dataR7X = 16'b1111\_1111\_1111\_1111;
82. //IO接口
83. portAddr = 16'b0000\_0000\_1111\_1111; //设定这里为port地址
84. portData = 16'b1111\_1111\_1111\_1111; //设定为port数据
85. end
86. always@(posedge clk)
87. begin
88. if(en == 1 && MemIO == 0)
89. begin
90. //从主存中读R
91. if(RW == 0)
92. begin
93. if(addr == addr0)
94. data\_read = data0;
95. else if(addr == addr1)
96. data\_read = data1;
97. else if(addr == addr2)
98. data\_read = data2;
99. else if(addr == addr3)
100. data\_read = data3;
101. else if(addr == addr4)
102. data\_read = data4;
103. else if(addr == addr5)
104. data\_read = data5;
105. else if(addr == addr6)
106. data\_read = data6;
107. else if(addr == addr7)
108. data\_read = data7;
109. else if(addr == addr8)
110. data\_read = data8;
111. else if(addr == addr9)
112. data\_read = data9;
113. else if(addr == addrR7X)
114. data\_read = dataR7X;
115. end
116. //从主存中写W
117. if(RW == 1)
118. begin
119. if(addr == addr1)
120. data1 = data\_write;
121. else if(addr == addr2)
122. data2 = data\_write;
123. else if(addr == addr3)
124. data3 = data\_write;
125. else if(addr == addr4)
126. data4 = data\_write;
127. else if(addr == addr5)
128. data5 = data\_write;
129. else if(addr == addr6)
130. data6 = data\_write;
131. else if(addr == addr7)
132. data7 = data\_write;
133. else if(addr == addr8)
134. data8 = data\_write;
135. else if(addr == addr9)
136. data9 = data\_write;
137. else if(addr == addrR7X)
138. dataR7X = data\_write;
139. end
140. end
141. if(en == 1 && MemIO == 1)
142. begin
143. //从IO中读R
144. if(RW == 0)
145. begin
146. if(addr == portAddr)
147. data\_read = portData;
148. end
149. //从IO中写W
150. if(RW == 1)
151. begin
152. if(addr == addr1)
153. portData = data\_write;
154. end
155. end
156. end
157. endmodule

### 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
19. T0  <= 1'b1;
20. T1  <= 1'b0;
21. T2  <= 1'b0;
22. T3  <= 1'b0;
23. end
24. if(count == 2'b01)
25. begin
26. T0  <= 1'b0;
27. T1  <= 1'b1;
28. T2  <= 1'b0;
29. T3  <= 1'b0;
30. end
31. if(count == 2'b10)
32. begin
33. T0  <= 1'b0;
34. T1  <= 1'b0;
35. T2  <= 1'b1;
36. T3  <= 1'b0;
37. end
38. if(count == 2'b11)
39. begin
40. T0  <= 1'b0;
41. T1  <= 1'b0;
42. T2  <= 1'b0;
43. T3  <= 1'b1;
44. end
45. if(reset)  *//清零信号=1，则清零*
46. count<=2'b00; *//有效计数器清零*
47. else
48. if(count == 2'b11) *//是否计导第N个数(0-N)*
49. count<=2'b00; *//清零*
50. else   *//否则加1*
51. count<=count+1;
52. end
53. endmodule

### Fetch.v

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2022/11/08 18:25:33*
7. *// Design Name:*
8. *// Module Name: Fetch*
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. module Fetch(
22. input clk,
23. input wire T0,
24. input wire T1,
25. output reg [15:0] PC\_out,*//往外输出的PC*
26. output reg [15:0] IR
27. );
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] PC;
33. wire [15:0] IR\_data;*//Mem找到之后给的指令数据*
35. CPU\_export cpu\_export(
36. .clk(clk),
37. .RW(RW),
38. .en(en),
39. .MemIO(MemIO),
40. .data\_write(data\_write),
41. .addr(PC),
42. .data\_read(IR\_data)
43. );
44. initial
45. begin
46. PC = 16'b0000\_0000\_0000\_0001;*//默认PC*
47. en =1'b0;   *//默认关闭*
48. end
49. always @(posedge T0)
50. begin
51. RW =1'b0;
52. en =1'b1;
53. MemIO =1'b0;
54. PC\_out <= PC;
55. end
56. always @(posedge T1)
57. begin
58. IR = IR\_data;
59. end
61. endmodule

### Execute.v

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2022/11/08 16:09:38*
7. *// Design Name:*
8. *// Module Name: Execute*
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. module Execute(
22. input T1,
23. input wire [7:0] valA,
24. input wire [7:0] valB,
25. input wire [7:0] X,
26. input wire [15:0] Addr,
27. input wire [15:0] IR,
28. output reg [7:0] ALUOUT
29. );
30. reg [4:0] operator;
31. always @(posedge T1)
32. begin
33. operator[4:0] = IR[15:11];
34. if(operator == 5'b00000)*//ADD*
35. begin
36. ALUOUT = valA + valB;
37. end
38. if(operator == 5'b00001)*//SUB*
39. begin
40. ALUOUT = valA - valB;
41. end
42. if(operator == 5'b00010)*// MOV*
43. begin
44. ALUOUT = valB;
45. end
46. if(operator == 5'b00011)*//MVI*
47. begin
48. ALUOUT = X;
49. end
50. if(operator == 5'b00100)*//STA*
51. begin
52. ALUOUT = valA;
53. end
54. if(operator == 5'b00101)*//LDA*
55. begin
57. end
58. if(operator == 5'b00110)*//JZ*
59. begin
60. ALUOUT = valA;
61. end
62. if(operator == 5'b00111)*//JMP*
63. begin
65. end
66. if(operator == 5'b01000)*//IN*
67. begin
68. end
69. if(operator == 5'b01001)*//OUT*
70. begin
71. ALUOUT = valA;
72. end
73. end
75. endmodule

### 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. module Access(
22. input clk,
23. input wire T2,
24. input wire [7:0] X,
25. input wire [15:0] IR,
26. input wire [15:0] Addr,
27. input wire [7:0] ALUOUT,
28. output reg [7:0] Rtemp
29. );
30. reg [4:0] operator;
31. reg RW;      *//0->R,1->W*
32. reg en;      *//en =1 代表可以*
33. reg MemIO;*//这次是访问IO还是Mem,0->mem,1->IO*
34. reg [15:0] data\_write;
35. reg [15:0] addr;
36. wire [15:0] data\_read;
37. CPU\_export cpu\_export(
38. .clk(clk),
39. .RW(RW),
40. .en(en),
41. .MemIO(MemIO),
42. .data\_write(data\_write),
43. .addr(addr),
44. .data\_read(data\_read)
45. );
46. always @(posedge T2)
47. begin
48. operator[4:0] = IR[15:11];
49. if(operator == 5'b00100)*//STA*
50. begin
51. *//模拟向主存写回*
52. *//ALUOUT->M(Addr)*
53. *//ALUOUT->M(Addr)*
54. *//1->W*
55. *//1->Mem*
56. RW = 1;
57. MemIO = 0;
58. en=1;
59. addr = Addr;
60. data\_write = ALUOUT;
61. end
62. if(operator == 5'b00101)*//LDA*
63. begin
64. *//模拟向主存读入*
65. RW = 0;
66. MemIO = 0;
67. en=1;
68. addr = Addr;
69. Rtemp = data\_read;
70. end
71. if(operator == 5'b01000)*//IN*
72. begin
73. *//从IO读入*
74. RW = 0;
75. MemIO = 1;
76. en=1;
77. *//需要拼接*
78. addr = {8'b0000\_0000,X};
79. Rtemp = data\_read;
80. end
81. if(operator == 5'b01001)*//OUT*
82. begin
83. *//从IO写入*
84. RW = 1;
85. MemIO = 1;
86. en=1;
87. *//需要拼接*
88. addr = {8'b0000\_0000,X};
89. data\_write = ALUOUT;
90. end
91. end
92. *//访存阶段*
93. endmodule

### 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. module WriteBack(
22. input wire T3,
23. input wire [7:0] Rtemp,
24. input wire [7:0] ALUOUT,
25. input wire [15:0] Addr,
26. input wire [15:0] IR,
27. output reg [15:0] PC\_jump\_data,
28. output reg  PC\_jump\_en,
29. output reg [7:0] R\_data,
30. output reg [15:0] R\_select,
31. output reg  R\_en
32. );
33. reg [4:0] operator;
34. always @(posedge T3)
35. begin
36. operator[4:0] = IR[15:11];
37. *//写回阶段*
38. *//执行ALUOUT -> Reg(Ad1(IR))*
39. if(operator == 5'b00000
40. || operator == 5'b00001
41. || operator == 5'b00010
42. || operator == 5'b00011)*//ADD SUB MOV MVI*
43. begin
44. R\_select = IR[10:8];
45. R\_data = ALUOUT;
46. R\_en = 1;
47. end
48. *//Rtemp -> Reg(Ad1(IR))*
49. if(operator == 5'b00101 || operator == 5'b01000) *//LDA IN*
50. begin
51. R\_select =  IR[10:8];
52. R\_data  = Rtemp;
53. R\_en =1;
54. end
55. *//If ALUOUT==0*
56. *// Then*
57. *// Addr->PC*
58. if(operator == 5'b00110) *//JZ*
59. begin
60. if(ALUOUT == 1'd0)
61. begin
62. PC\_jump\_data = Addr;
63. PC\_jump\_en = 1;
64. end
65. end
66. *//Addr->PC*
67. if(operator == 5'b00111)*//JMP*
68. begin
69. PC\_jump\_data = Addr;
70. PC\_jump\_en = 1;
71. end
72. if(operator == 5'b01001)*//OUT*
73. begin
74. end
75. end
76. endmodule

### 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. 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
23. 成功
24. \*/
25. module CPU\_export\_read();
26. reg clk;
27. reg RW;      *//0->R,1->W*
28. reg en;      *//en =1 代表可以*
29. reg MemIO;*//这次是访问IO还是Mem,0->mem,1->IO*
30. reg [15:0] data\_write;
31. reg [15:0] addr;
32. wire [15:0] data\_read;
33. always #5 clk=~clk;*//每10ns一次*
34. initial
35. begin
36. clk = 0;
37. RW=0;
38. en=1;
39. MemIO=0;
40. addr =16'b0000\_0000\_0000\_0000;
41. end
42. CPU\_export cpu\_export\_readTest(
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. endmodule

### CPU\_export\_write.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. 测试CPU接口模块，
23. 输入addr 0000\_0000\_0000\_0000
24. 向该地址写入11111\_111\_111\_11111
25. 然后查看addr 0000\_0000\_0000\_0000
26. 发现已经成为11111\_111\_111\_11111
27. 成功
28. \*/
29. module CPU\_export\_write();
30. reg clk;
31. reg RW;      *//0->R,1->W*
32. reg en;      *//en =1 代表可以*
33. reg MemIO;*//这次是访问IO还是Mem,0->mem,1->IO*
34. reg [15:0] data\_write;
35. reg [15:0] addr;
36. wire [15:0] data\_read;
37. always #5 clk=~clk;*//每10ns一次*
38. initial
39. begin
40. *//先写*
41. clk = 0;
42. RW=1;
43. en=1;
44. MemIO=0;
45. addr =16'b0000\_0000\_0000\_0000;
46. data\_write =16'b11111\_111\_111\_11111;
47. *//后通过data\_read看*
48. #20
49. RW=0;
50. en=1;
51. MemIO=0;
52. addr =16'b0000\_0000\_0000\_0000;
53. end
54. CPU\_export cpu\_export\_readTest(
55. .clk(clk),
56. .RW(RW),
57. .en(en),
58. .MemIO(MemIO),
59. .data\_write(data\_write),
60. .addr(addr),
61. .data\_read(data\_read)
62. );
63. endmodule

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

### 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. 测试取指模块
23. PC为默认16'b0000\_0000\_0000\_0000
24. 取出IR指令00000\_000\_001\_00000
25. 成功
26. \*/
27. module Fetch\_tb\_readPC();
28. reg clk;
29. reg T0;
30. reg T1;
31. wire [15:0] PC\_out;*//往外输出的PC*
32. wire [15:0] IR;
33. always #5 clk=~clk;//每10ns一次
34. initial
35. begin
36. clk=0;
37. T0=0;
38. T1=0;
39. #20 T0=~T0;
40. #20 T1=~T1;
41. end
42. Fetch fetch\_test\_defalut(
43. .clk(clk),
44. .T0(T0),
45. .T1(T1),
46. .PC\_out(PC\_out),
47. .IR(IR)
48. );
49. endmodule

### Decode\_tb.v

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2022/11/08 16:37:23*
7. *// Design Name:*
8. *// Module Name: Decode\_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. 测试译码模块，输入IR 00000\_000\_001\_00000，
23. 译码得到输出valA，valB，X，Addr
24. 成功
25. \*/
26. module Decode\_tb\_add();
27. reg T0;
28. reg [15:0] IR;
29. wire [7:0] valA;
30. wire [7:0] valB;
31. wire [7:0] X;
32. wire [15:0] Addr;
33. always #5 T0=~T0;*//每10ns一次*
34. initial
35. begin
36. T0 = 1'b0;
37. IR = 16'b00000\_000\_001\_00000;*//测试ADD R0+R1*
38. end
39. Decode testADD(T0,IR,valA,valB,X,Addr);
40. endmodule

### Execute\_tb.v

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2022/11/08 16:14:40*
7. *// Design Name:*
8. *// Module Name: Execute\_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. 测试执行模块，
23. 输入IR 00000\_000\_001\_00000，valA，valB，X，Addr
24. 译码得到输出ALUOUT
25. 成功
26. \*/
27. module Execute\_tb\_add();
28. reg T1;
29. reg [7:0] valA;
30. reg [7:0] valB;
31. reg [7:0] X;
32. reg [15:0] Addr;
33. reg [15:0] IR;
34. wire [7:0] ALUOUT;
35. always #5 T1=~T1;*//每10ns一次*
36. initial
37. begin
38. T1 = 1'b0;
39. valA = 8'b0000\_0001;
40. valB = 8'b0000\_1110;
41. X    = 8'b0000\_1111;
42. Addr = 16'b0000\_0000\_0000\_0001;
43. IR = 16'b00000\_000\_001\_00000;*//测试ADD R0+R1（valA+valB）*
44. end
45. Execute testADD(T1,valA,valB,X,Addr,IR,ALUOUT);
46. endmodule