计算机体系结构实验课程第__一__次实报告

实验名称	多周期 CPU 实现			班级	李雨森老师
学生姓名	蒋薇	学号	2110957	指导老师	董前琨
实验地点	A308		实验时间	2023.09.25	

1、 实验目的

1. 在单周期 CPU 实验完成的提前下，理解多周期的概念。
2. 熟悉并掌握多周期 CPU 的原理和设计。
3. 进一步提升运用 verilog 语言进行电路设计的能力。
4. 为后续实现流水线 cpu 的课程设计打下基础。

2、 实验内容说明

1. 做好预习：

1） 复习单周期 CPU 的实验内容，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令，对其进行分析，完成表 8.1 的填写；

2） 依据自己设计中实现的指令，编写一段不少于 40 行的汇编程序，要求包含所有实现的指令，完成表 8.2 的填写；

3） 认真学习多周期的概念，了解流水线的概念，明白划分为多周期的意义；

4） 认真学习 CPU 各模块的功能，确认模块的划分。设计本次实验的方案，画出实验方案的设计框图，即补充完善图 8.1；

5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 8.2。

2. 实验实施：

1） 确认多周期 CPU 的设计框图的正确性；

2） 编写 verilog 代码，将表 8.2 中自己编写的汇编程序翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；

3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料，在仿真时需要将生成指令 ROM 时产生的.mif 文件拷贝到工程目录下，才能仿真成功；

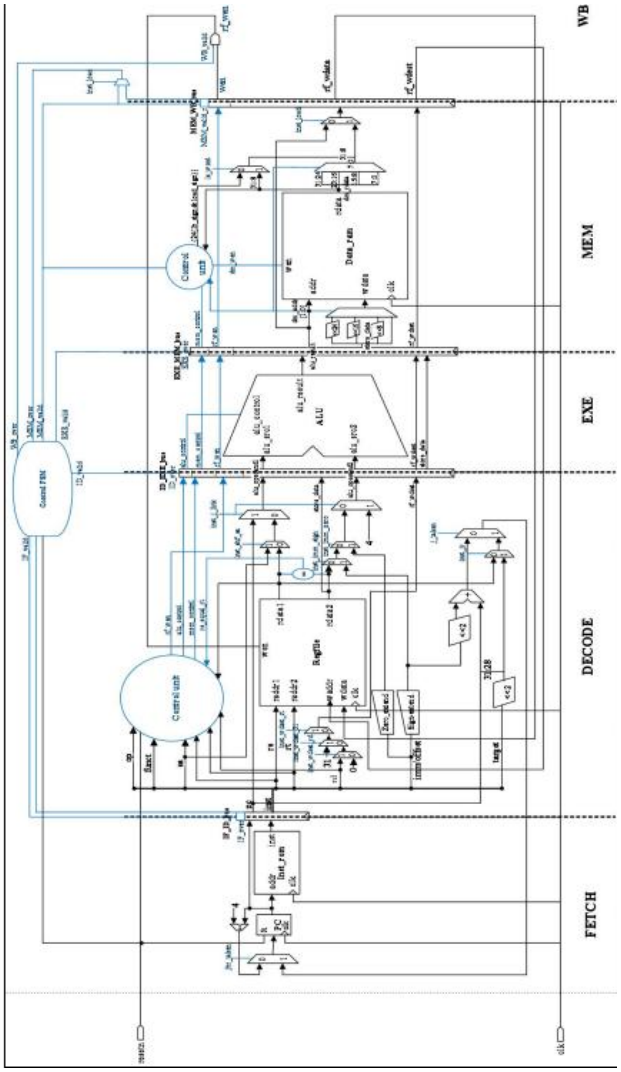
4） 完成调用多周期 CPU 的外围模块的设计，并编写代码；

5） 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

将实验六所实现的单周期 CPU 划分为多周期的，并扩展指令到 30 多条,ROM 和 RAM 使用调用库 IP 实例化的同步存储器。

3、 实验原理图

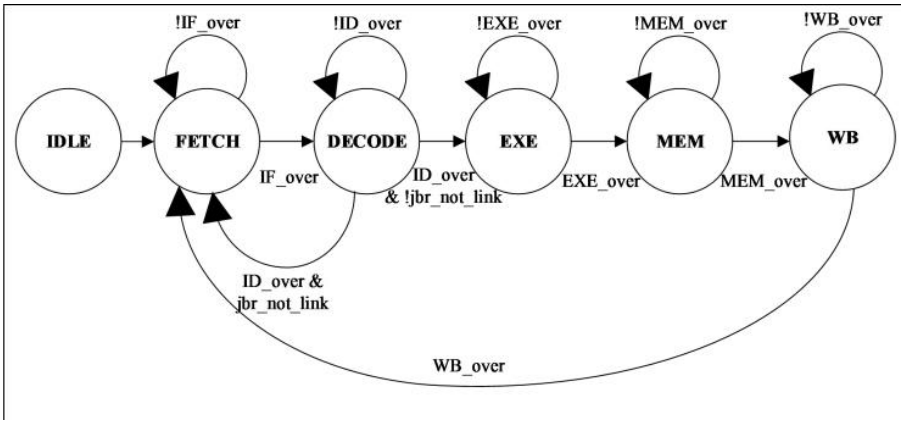
下图为多周期 cpu 实现框图：



多周期 CPU 实现的指令集是在单周期设计的基础上增加 20 条算术逻辑运算指令、 数据传送指令和控制指令，共实现了 36 条指令。

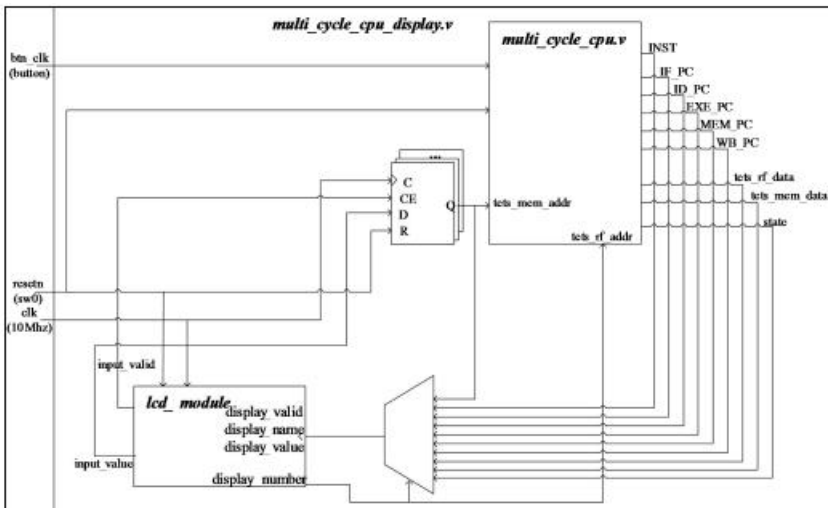
多周期 CPU 设计在单周期 CPU 基础上，主要做两部分改进。第一部分是控制单元，增加控制电路使每一个时钟只有一个阶段的电路产生的结果有效，并锁存上一阶 段的结果用于后续阶段的运行；第二部分是数据通路，增加实现新增指令的电路。

第一部分的改进主要是增加状态机控制及增加各阶段之间的用于锁存的寄存器。由于有 5 个阶段，状态机共有 6 个状态：空闲（IDLE）、取指（FETCH）、译码 （DECODE）、执行（EXE）、访存（MEM）、写回（WB），



空闲（IDLE）状态：CPU 在复位时，所有阶段电路都无效，CPU 等待复位结束开始下一状态——取指。
取指（FETCH）状态：在状态机进入取指状态的同时，PC 更新为下一 PC 值。故取 指状态下，将 PC 值作为指令存储器的地址去取指令。由于同步指令存储器在下一时钟周期返回指令，因此取指需要两个时钟周期的时间。当取指结束后锁存取指阶段产生的结果——当前 PC 值和指令。
译码（DECODE）状态：类似于单周期 CPU 的译码阶段，主要完成指令译码、读寄存器、判断跳转等。控制单元区分各条指令并产生用于译码、执行、访存、写回的控制信号。当译码结束后锁存译码阶段产生的结果用于下一状态执行：分别用于执行访存、写回的控制信号、用于执行阶段的两个源操作数、用于访存阶段的写入内存数据、用于写回阶段的写寄存器地址。
执行（EXE）状态：ALU 模块完成操作。当执行结束后锁存执行阶段产生的结果及前级传递的结果：用于访存、写回的控制信号、ALU 结果、内存写入数据、寄存器写地址。
访存（MEM）状态：完成对数据存储器的读或写，并选择出将要写回寄存器的值。当访存结束后锁存访存阶段产生的结果及前级传递的结果：用于写回的控制信号、写回数据、写回地址。
写回（WB）状态：完成寄存器写入。
CPU 复位结束，状态机由 IDLE 进入取指状态，其后在每次上一级结束信号有效的时进入下一状态，写回级结束后返回取指级。当然也有例外，当指令是跳转而非链接跳转指令时，在译码状态后直接返回取下一指令不需要经过执行等后续阶段。

下图为多周期实验顶层模块框图：



4、 实验步骤

补充表 8.1，mips 基础指令特性归纳表：

指令类型	汇编指令	指令码	源操作数 1	源操作数 2	源操作数 3	目的寄存器	功能描述
R 型指令	addu rd,rs,rt	000000|rs|rt|rd|00000|100001	[rs]	[rt]		rd	GPR[rd]=GPR[rs]+GPR[rt]
	subu rd,rs,rt	000000|rs|rt|rd|00000|100011	[rs]	[rt]		rd	GPR[rd]=GPR[rs]-GPR[rt]
	slt rd,rs,rt	000000|rs|rt|rd|00000|101010	[rs]	[rt]		rd	GPR[rd]=(sign(GPR[rs])<sign(GPR[rt]))
	sltu rd,rs,rt	000000|rs|rt|rd|00000|101011	[rs]	[rt]		rd	GPR[rd]=(zero(GPR[rs])<zero(GPR[rt]))
	jalr rs	000000|rs|00000|11111|00000|001001	[rs]			31	GPR[31]=PC,PC=GPR[rs]
	jr rs	000000|rs|000000000000|00000|001000	[rs]				PC=GPR[rs]

R 型指令	and rd,rs,rt	000000|rs|rt|rd|00000|100100	[rs]	[rt]		rd	GPR[rd]=GPR[rs]&GPR[rt]
	nor rd,rs,rt	000000|rs|rt|rd|00000|100111	[rs]	[rt]		rd	GPR[rd]=~(GPR[rs]|GPR[rt])
	or rd,rs,rt	000000|rs|rt|rd|00000|100101	[rs]	[rt]		rd	GPR[rd]=GPR[rs]|GPR[rt]
	xor rd,rs,rt	000000|rs|rt|rd|00000|100110	[rs]	[rt]		rd	GPR[rd]=GPR[rs]^GPR[rt]
	sll rd,rt,shf	000000|00000|rt|rd|shf|000000		[rt]		rd	GPR[rd]=zero(GPR[rt])<<shf
	sllv rd,rt,rs	000000|rs|rt|rd|00000|000100	[rs]	[rt]		rd	GPR[rd]=zero(GPR[rt])<<(GPR[rs]%32)
	sra rd,rt,shf	000000|00000|rt|rd|shf|000011		[rt]		rd	GPR[rd]=sign(GPR[rt])>>shf
	srav rd,rt,rs	000000|rs|rt|rd|00000|000111	[rs]	[rt]		rd	GPR[rd]=sign(GPR[rt])>>(GPR[rs]%32)
	srl rd,rt,shf	000000|00000|rt|rd|shf|000010		[rt]		rd	GPR[rd]=zero(GPR[rt])>>shf
	srlv rd,rt,rs	000000|rs|rt|rd|00000|000110	[rs]	[rt]		rd	GPR[rd]=zero(GPR[rt])>>GPR[rs]
I 型指令	addiu rt,rs,imm	001001|rs|rt|imm	[rs]	sign_ext (imm)		rt	GPR[rt]=GPR[rs]+ sign_ext (imm)
	slti rt,rs,imm	001010|rs|rt|imm	[rs]	sign_ext (imm)		rt	GPR[rt]=(sign(GPR[rs])< sign_ext (imm))
	sltiu rt,rs,imm	001011|rs|rt|imm	[rs]	sign_ext (imm)		rt	GPR[rt]=(zero(GPR[rs])< sign_ext (imm))
	beq rs,rt,offset	000100|rs|rt|offset	[rs]	[rt]			if GPR[rs]=GPR[rt] then PC=PC+ sign_ext (offset)<<2
	bgez rs,offset	000001|rs|00001|offset	[rs]				if GPR[rs]≥0 then PC=PC+ sign_ext (offset)<<2
	bgtz rs,offset	000111|rs|00000|offset	[rs]				if GPR[rs]>0 then PC=PC+ sign_ext (offset)<<2
	blez rs,offset	000110|rs|00000|offset	[rs]				if GPR[rs]≤0 then PC=PC+ sign_ext (offset)<<2
	bltz rs,offset	000001|rs|00000|offset	[rs]				if GPR[rs]<0 thenPC=PC+ sign_ext (offset)<<2
	bne rs,rt,offset	000101|rs|rt|offset	[rs]	[rt]			if GPR[rs]≠GPR[rt] then PC=PC+ sign_ext (offset)<<2
	lw rt,offset(b)	100011|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=Mem[GPR[b]+sign_ext (offset)]
	sw rt,offset(b)	101011|b|rt|offset	[b]	sign_ext (offset)	[rt]		Mem[GPR[b]+sign_ext (offset)]=GPR[rt]
	lb rt,offset(b)	100000|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=sign(Mem[GPR[b]+sign_ext (offset)])
	lbu rt,offset(b)	100100|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=zero(Mem[GPR[b]+sign_ext (offset)])
	sb rt,offset(b)	101000|b|rt|offset	[b]	sign_ext (offset)	[rt]		Mem[GPR[b]+sign_ext(offset)]=GPR[rt]
	andi rt,rs,imm	001100|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]&zero_ext (imm)
	lui rt,imm	001111|00000|rt|imm		{imm, 16'd0}		rt	GPR[rt]= {imm, 16'd0}
	ori rt,rs,imm	001101|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]|zero_ext (imm)
I 型指令	xori rt,rs,imm	001110|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]^zero_ext (imm)
J 型指令	j target	000010|target					PC={PC[31:28],target<<2}
	jal target	000011|target					GPR[31]=PC,PC={PC[31:28],target<<2}

补充表 8.2，多周期 CPU 测试所用汇编程序详述：

指令地址	汇编指令	结果描述	机器指令的机器码	
			16 进制	二进制
00H	addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001
04H	sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000
08H	addu $3, $2,$1	[$3] = 0000_0011H	00411821	0000_0000_0100_0001_0001_1000_0010_0001
0CH	srl $4, $2,#2	[$4] = 0000_0004H	00022082	0000_0000_0000_0010_0010_0000_1000_0010
10H	slti $25,$4,#5	[$25] = 0000_0001H	28990005	0010_1000_1001_1001_0000_0000_0000_0101
14H	bgez $25,#16	跳转到 54H	07210010	0000_0111_0010_0001_0000_0000_0001_0000
18H	subu $5, $3,$4	[$5] = 0000_000DH	00642823	0000_0000_0110_0100_0010_1000_0010_0011
1CH	sw $5, #20($0)	Mem[0000_0014H] = 0000_000DH	AC050014	1010_1100_0000_0101_0000_0000_0001_0100
20H	nor $6, $5,$2	[$6] = FFFF_FFE2H	00A23027	0000_0000_1010_0010_0011_0000_0010_0111
24H	or $7, $6,$3	[$7] = FFFF_FFF3H	00C33825	0000_0000_1100_0011_0011_1000_0010_0101
28H	xor $8, $7,$6	[$8] = 0000_0011H	00E64026	0000_0000_1110_0110_0100_0000_0010_0110
2CH	sw $8, #28($0)	Mem[0000_001CH] = 0000_0011H	AC08001C	1010_1100_0000_1000_0000_0000_0001_1100
30H	beq $8, $3,#2	跳转到 38H	11030002	0001_0001_0000_0011_0000_0000_0000_0010
34H	slt $9, $6,$7	不执行	00C7482A	0000_0000_1100_0111_0100_1000_0010_1010
38H	addiu $1, $0,#8	[$1] = 0000_0008H	24010008	0010_0100_0000_0001_0000_0000_0000_1000
3CH	lw $10,#20($1)	[$10] = 0000_0011H	8C2A0014	1000_1100_0010_1010_0000_0000_0001_0100
40H	bne $10,$5,#4	跳转到 50H	15450004	0001_0101_0100_0101_0000_0000_0000_0100
44H	and $11,$2,$1	不执行	00415824	0000_0000_0100_0001_0101_1000_0010_0100
48H	sw $11,#28($1)	不执行	AC2B001C	1010_1100_0010_1011_0000_0000_0001_1100
4CH	sw $4, #16($1)	不执行	AC240010	1010_1100_0010_0100_0000_0000_0001_0000
50H	jal #25	跳转到 64H, [$31] = 0000_0054H	0C000019	0000_1100_0000_0000_0000_0000_0001_1001
54H	lui $12,#12	[$12] = 000C_0000H	3C0C000C	0011_1100_0000_1100_0000_0000_0000_1100
58H	srav $26,$12,$2	[$26] = 0000_000CH	004CD007	0000_0000_0100_1100_1101_0000_0000_0111
5CH	sllv $27,$26,$1	[$27] = 0000_0018H	003AD804	0000_0000_0011_1010_1101_1000_0000_0100
60H	jalr $27	跳转到 18H , [$31] = 0000_0064H	0360F809	0000_0011_0110_0000_1111_1000_0000_1001
64H	sb $26,#5($3)	MEM[0000_0016H] = 000C_000DH	A07A0005	1010_0000_0111_1010_0000_0000_0000_0101
68H	sltu $13,$3,$3	[$13] = 0000_0000H	0063682B	0000_0000_0110_0011_0110_1000_0010_1011
6CH	bgtz $13,#3	不跳转	1DA00003	0001_1101_1010_0000_0000_0000_0000_0011
70H	sllv $14,$6,$4	[$14] =FFFF_FE20H	00867004	0000_0000_1000_0110_0111_0000_0000_0100
74H	sra $15,$14,#2	[$15] =FFFF_FF88H	000E7883	0000_0000_0000_1110_0111_1000_1000_0011
78H	srlv $16,$15,$1	[$16] =00FF_FFFFH	002F8006	0000_0000_0010_1111_1000_0000_0000_0110
7CH	blez $16,#8	不跳转	1A000008	0001_1010_0000_0000_0000_0000_0000_1000
80H	srav $16,$15,$1	[$16] =FFFF_FFFFH	002F8007	0000_0000_0010_1111_1000_0000_0000_0111
84H	addiu $11,$0,#140	[$11] = 0000_008CH	240B008C	0010_0100_0000_1011_0000_0000_1000_1100
88H	bltz $16, #6	跳转到 A0H	06000006	0000_0110_0000_0000_0000_0000_0000_0110
8CH	lw $28,#3($10)	[$28] = 000C_000DH /000C_880DH	8D5C0003	1000_1101_0101_1100_0000_0000_0000_0011
90H	bne $28,$29,#7	不跳转/跳转 ACH	179D0007	0001_0111_1001_1101_0000_0000_0000_0111
94H	sb $15,#8($5)	Mem[0000_0015H] = 0000_0088H	A0AF0008	1010_0000_1010_1111_0000_0000_0000_1000
98H	lb $18,#8($5)	[$18] =FFFF_FF88H	80B20008	1000_0000_1011_0010_0000_0000_0000_1000
9CH	lbu $19,#8($5)	[$19] = 0000_0088H	90B30008	1001_0000_1011_0011_0000_0000_0000_1000
A0H	sltiu $24,$15,#0xFFFF	[$24] = 0000_0001H	2DF8FFFF	0010_1101_1111_1000_1111_1111_1111_1111
A4H	or $29,$12,$5	[$29] = 000C000DH	0185E825	0000_0001_1000_0101_1110_1000_0010_0101
A8H	jr $11	跳转指令 8CH	01600008	0000_0001_0110_0000_0000_0000_0000_1000
ACH	andi $20,$15,#0xFFFF	[$20] = 0000_FF88H	31F4FFFF	0011_0001_1111_0100_1111_1111_1111_1111
B0H	ori $21,$15,#0xFFFF	[$21] =FFFF_FFFFH	35F5FFFF	0011_0101_1111_0101_1111_1111_1111_1111
B4H	xori $22,$15,#0xFFFF	[$22] = FFFF_0077H	39F6FFFF	0011_1001_1111_0110_1111_1111_1111_1111
B8H	j #00H	跳转指令 00H	08000000	0000_1000_0000_0000_0000_0000_0000_0000

5、 实验结果分析

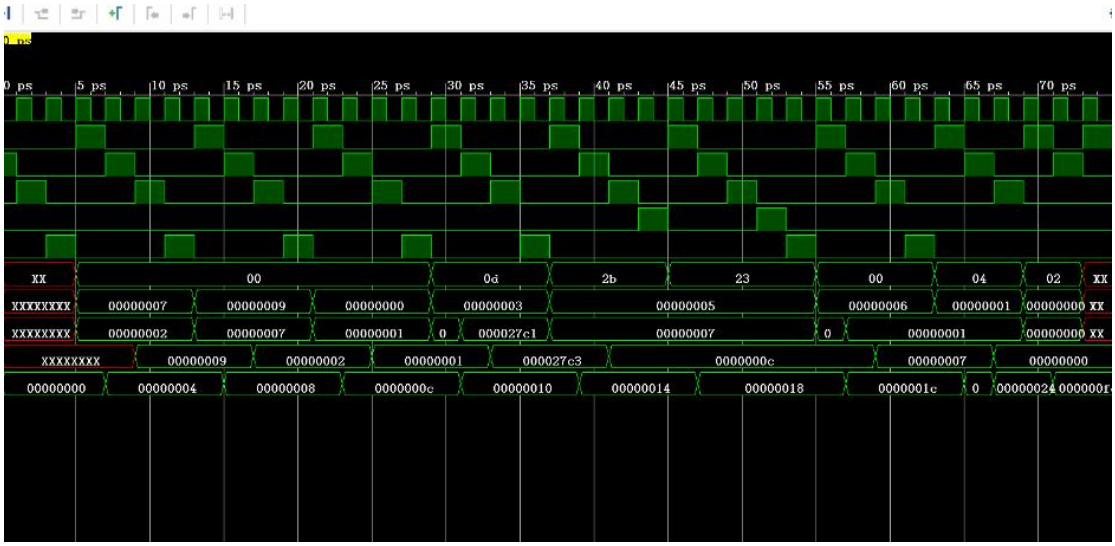
我们需要在每一个时钟里查看一条指令的运算结果，故演示时理想的时钟是手动输入的，可以使用 FPGA 板上的脉冲开关代替时钟。

触摸屏上共有 44 块显示区域，可显示 44 组 32 位数据 ，44 块显示区域从 1 开始编号，编号为 1~44，块号 5~36 显示 32 个通用寄存器的值。

数据值使用十六进制表示，存储器的第一个地址的数据值为 0x24010001，第二个地址的数据值为 0x00011100，第三个地址的数据值为 0x00411821，以此类推。

单周期 CPU 实验采用的是异步存储器（data_ram.v 和 inst_rom.v），多周期实验中因为又时钟控制周期，需要使用同步存储器（新建 IP 核的形式，参考存储器实验）；

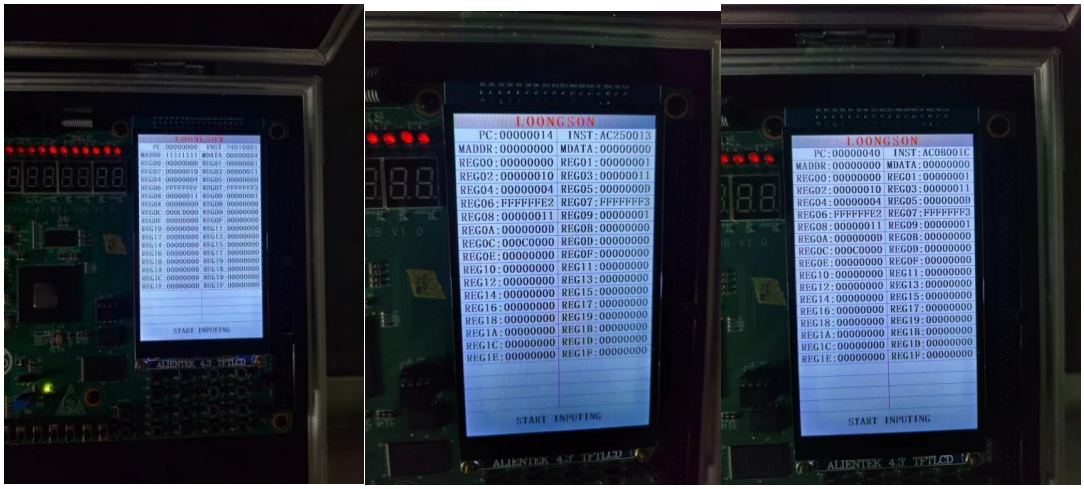
仿真：对照 CLK、IF_clk、ID_clk、ALU_clk、MEM_clk、WB_clk 所划分的周期可以很容易验证指令的执行情况。



I 型指令：addiu

00H	addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001

分析：加法操作指令，将 0 号寄存器的数据（0）与立即数 1 相加，并且将运算结果（1）写回 1 号寄存器。需要取指令，指令译码，指令执行，结果写回四个步骤，不需要存储器访问，因此四个周期即可完成，在最后一个时钟周期的下降沿写入数据。

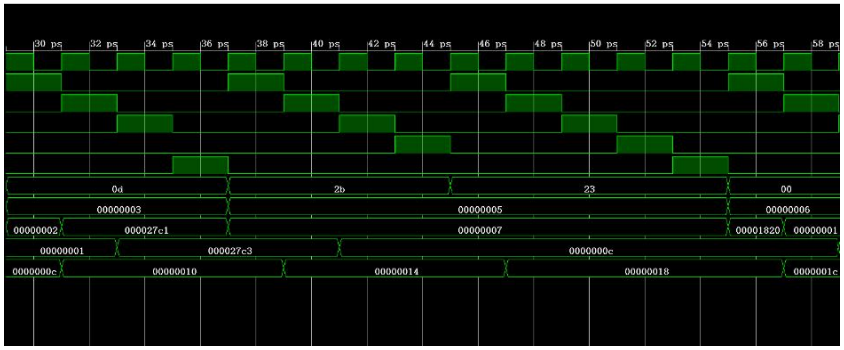


R 型指令：sll

04H	sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000

分析：移位操作指令，将 1 号寄存器的数据（2）左移 4 位，并且将运算结果（10H）写回 2 号寄存器。需要取指令，指令译码，指令执行，结果写回四个步骤，不需要存储器访问，因此四个周期即可完成，在最后一个时钟周期的下降沿写入数据。

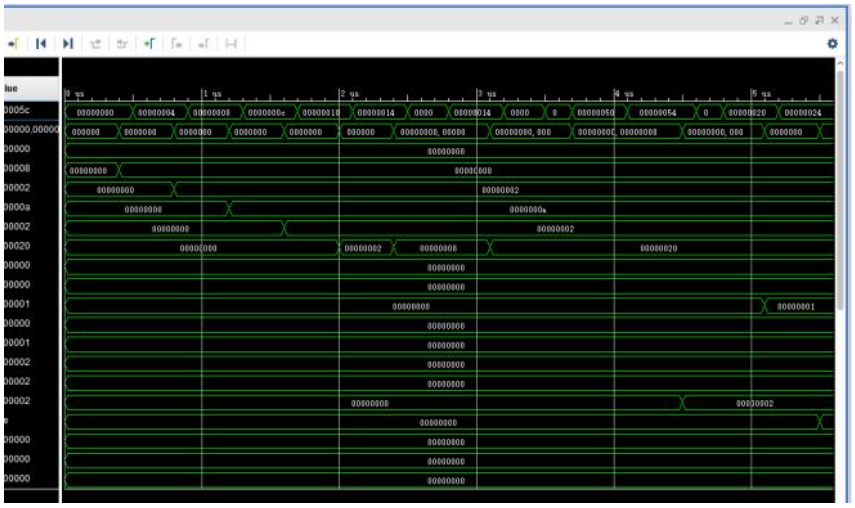
执行地址 0x00000004 的时候，执行了 sll $2,$1,#4,使得[$2] = 0000 0010H，继续执行下一条指令...



J 型指令：j

B8H	j #00H	跳转指令 00H	08000000	0000_1000_0000_0000_0000_0000_0000_0000

分析：跳转操作指令，执行地址 0x000000D8 的时候，执行了 j #00H,跳转指令 00H



MIPS 指令三种类型：

R类型：

31　26 25　21 20　16 15　11 10　6 5　0

op	rs	rt	rd	sa	func

6位　5位　5位　5位　5位　6位

I类型：

31　26 25　21 20　16 15　0

op	rs	rt	immediate

6位　5位　5位　16位

J类型：

31　26 25　0

op	address

6位　26位

算术运算指令，（1）add rd, rs, rt（2）sub rd, rs, rt（3）addi　rt, rs, **immediate**

逻辑运算指令（4）or rd, rs, rt（5）and rd, rs, rt（6）ori rt, rs, **immediate**

移位指令（7）sll rd, rs,sa

传送指令（8）move　rd, rs

比较指令（9）　slt rd, rs, rt

存储器读写指令（10）sw rt, **immediate(**rs)（11）lw rt, **immediate**(rs)

分支指令（12）beq rs,rt, **immediate**

跳转指令（13）j addr　　（14）jr rs

调用子程序指令（15）jal addr

停机指令（16）halt (停机指令)

6.总结感想

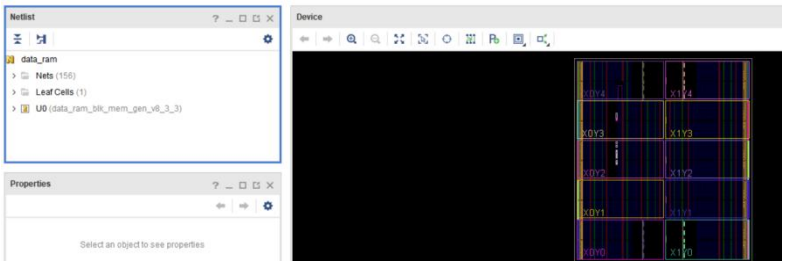
将指令代码初直接写入始化到指令存储器中。初始化 PC 的值，也就是以上程序段首地址 PC=0x00000000，以上程序段从 0x00000000 地址开始存放。运行仿真，看波形，使用表进行测试 CPU 正确性。

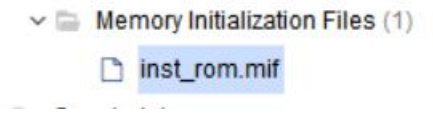
Run Simulation ;

Run Implementation;

[DRC INBB-3] Black Box Instances: Cell 'cpu/data_ram_module' of type 'cpu/data_ram_module/data_ram' has undefined contents and is considered a black box. The contents of this cell must be defined for opt_design to complete successfully.

表示在设计中存在一个黑盒实例。具体来说，‘cpu/data_ram_module’单元的类型是‘cpu/data_ram_module/data_ram’，但其内容未定义。为了使 opt_design 成功完成，必须定义此单元的内容。





mif 文件用于初始化存储器内容，

inst_rom.mif

```
1  00100100000000010000000000000001
2  00000000000000001000100010000000
3  00000000010000010001100000100001
4  00000000000000010001000001000010
5  00101000100110010000000000000101
6  00000111001000010000000000010000
7  00000000011001000010100000100011
8  10101100000001010000000000010100
9  00000000101000100011000000100111
10 00000000110000110011100000100101
11 00000000111001100100000000100110
12 10101100000010000000000000011100
13 00010001000000110000000000000010
14 00000000110001110100100000101010
15 00100100000000010000000000001000
16 10001100001010100000000000010100
17 00010101010001010000000000000100
```

按照其错误提示，结合 data_ram

[VRFC 10-2063] Module <BLK_MEM_GEN_V6_1> not found while processing module instance <inst>

["D:/junior/Computer_Architeture/source_code/source_code/7_multi_cycle_cpu/inst_rom.v":51]

表示在处理模块实例<inst>时找不到<BLK_MEM_GEN_V6_1>模块。具体来说，该错误发生在"D:/junior/Computer_Architeture/source_code/source_code/7_multi_cycle_cpu/inst_rom.v"文件的第51 行。

#号的问题

[DRC INBB-3] Black Box Instances: Cell 'cpu/inst_rom_module' of type 'cpu/inst_rom_module/inst_rom' has undefined contents and is considered a black box. The contents of this cell must be defined for opt_design to complete successfully.表示单元格'cpu/inst_rom_module'的类型为'cpu/inst_rom_module/inst_rom'的内容未定义，被视为黑箱。为了使_design 成功完成，必须定义该单元格的内容。

结合报错信息更改，

Run Simulation ;

Run Implementation;

Generate Bitstream 上箱

多周期 CPU 相对于单周期主要多思考自动状态机转换的问题，此外，还需要增加寄存器用于接收中间的结果，并且需要连上时钟。