实 验 三　多周期 CPU 设计与实现

：

（完成时间：十五、十六周）

一、 实验目的

(1) 认识和掌握多周期数据通路原理及其设计方法；
(2) 掌握多周期 CPU 的实现方法，代码实现方法；
(3) 编写一个编译器，将 MIPS 汇编程序编译为二进制机器码；
(4) 掌握多周期 CPU 的测试方法；
(5) 掌握多周期 CPU 的实现方法。

二、 实验内容

设计一个多周期 CPU，该 CPU 至少能实现以下指令功能操作。需设计的指令与格式如下：

（**说明：操作码按照以下规定使用，都给每类指令预留扩展空间，后续实验相同。**）

==>算术运算指令

（1）add　rd，rs，rt

000000	rs(5 位)	rt(5 位)	rd(5 位)	reserved

功能：rd<－rs + rt

（2）sub　rd，rs，rt

000001	rs(5 位)	rt(5 位)	rd(5 位)	reserved

完成功能：rd<－rs － rt

（3）addi　rt，rs，**immediate**

000010	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：rt<－rs + (sign-extend)**immediate**

==>逻辑运算指令

（4）or　rd，rs，rt

010000	rs(5 位)	rt(5 位)	rd(5 位)	reserved

功能：rd<－rs | rt

（5）and　rd，rs，rt

010001	rs(5 位)	rt(5 位)	rd(5 位)	reserved

功能：rd<－rs & rt

（6）ori　rt，rs，**immediate**

010010	rs(5 位)	rt(5 位)	**immediate**

功能：rt<－rs | (zero-extend)**immediate**

==>移位指令

（7）sll　rd，rt,sa

011000	未用	rt(5 位)	rd(5 位)	sa(5 位)	reserved

功能：rd<－rt<<(zero-extend)sa，左移 sa 位 ，(zero-extend)sa

==>比较指令

（8） slt　rd，rs，rt　带符号数

100110	rs(5 位)	rt(5 位)	rd(5 位)	reserved

功能：if (rs<rt) rd =1 else rd=0, 具体请看表 2 ALU 运算功能表，带符号

（9） slti rt, rs,immediate 带符号

100111	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：if (rs <(sign-extend)**immediate**) rt =1 else rt=0, 具体请看表 2 ALU 运算功能表，带符号

==>存储器读写指令

（10）sw rt, **immediate**(rs)

110000	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：memory[rs+ (sign-extend)**immediate**]<—rt。即将 rt 寄存器的内容保存到 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中。

（11）lw rt, **immediate**(rs)

110001	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：rt <— memory[rs + (sign-extend)**immediate**]。即读取 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后保存到 rt 寄存器中。

==>分支指令

（12）beq rs,rt, **immediate** (说明：**immediate 从 pc+4 开始和转移到的指令之间间隔条数**)

110100	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：if(rs=rt) pc <—pc + 4 + (sign-extend)**immediate** <<2 else pc <—pc + 4

（13）bne rs,rt, **immediate** (说明：**immediate 从 pc+4 开始和转移到的指令之间间隔条数**)

110101	rs(5 位)	rt(5 位)	**immediate**(16 位)

功能：if(rs!=rt) pc <—pc + 4 + (sign-extend)**immediate** <<2 else pc <—pc + 4

（14）bgtz rs,**immediate**

110110	rs(5 位)	00000	**immediate**

功能：if(rs>0) pc←pc + 4 + (sign-extend)**immediate** <<2 else pc ←pc + 4

==>跳转指令

（15）j addr

111000	addr[27:2]

功能：pc <—{(pc+4)[31:28],addr[27:2],0,0}，跳转。

说明：由于 MIPS32 的指令代码长度占 4 个字节，所以指令地址二进制数最低 2 位均为 0，将指令地址放进指令代码中时，可省掉！这样，除了最高 6 位操作码外，还有 26 位可用于存放地址，事实上，可存放 28 位地址，剩下最高 4 位由 pc+4 最高 4 位拼接上。

（16）jr rs

111001	rs(5 位)	未用	未用	reserved

功能：pc <— rs，跳转。

==>调用子程序指令

（17）jal addr

111010	addr[27..2]

功能：调用子程序，pc <— {(pc+4)[31:28],addr[27:2],0,0}；$31<—pc+4，返回地

址设置；子程序返回，需用指令 jr $31。跳转地址的形成同 j addr 指令。

==>停机指令

（18）halt（停机指令）

111111	00000000000000000000000000(26 位)

不改变 pc 的值，pc 保持不变。

在本文档中，提供的相关内容对于设计可能不足或甚至有错误，希望同学们在设计过程中如发现有问题，请你们自行改正，进一步补充、完善。谢谢！

三、 实验原理

多周期 CPU 指的是将整个 CPU 的执行过程分成几个阶段，每个阶段用一个时钟去完成，然后开始下一条指令的执行，而每种指令执行时所用的时钟数不尽相同，这就是所谓的多周期 CPU。CPU 在处理指令时，一般需要经过以下几个阶段：

(1) 取指令(IF)：根据程序计数器 pc 中的指令地址，从存储器中取出一条指令，同时，pc 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 pc，当然得到的“地址”需要做些变换才送入 pc。

(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

实验中就按照这五个阶段进行设计，这样一条指令的执行最长需要五个(小)时钟周期才能完成，但具体情况怎样？要根据该条指令的情况而定，有些指令不需要五个时钟周期的，这就是多周期的 CPU。

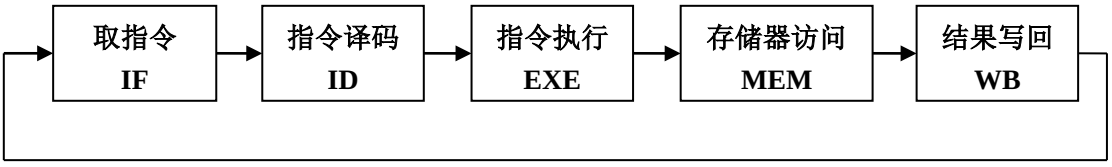
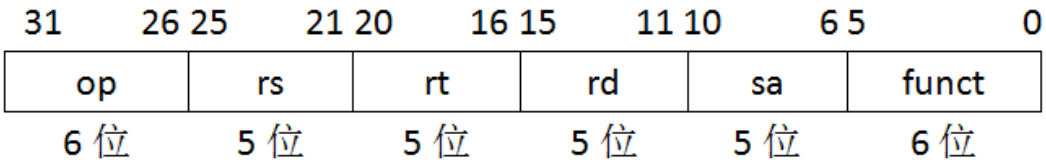


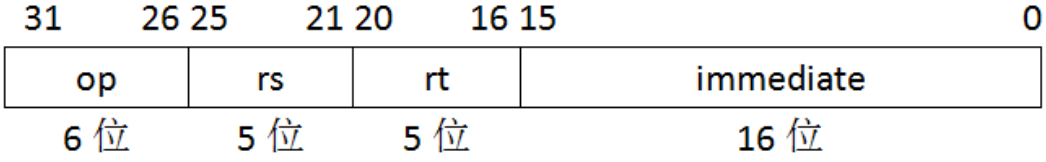
图 1　多周期 CPU 指令处理过程

MIPS 指令的三种格式：

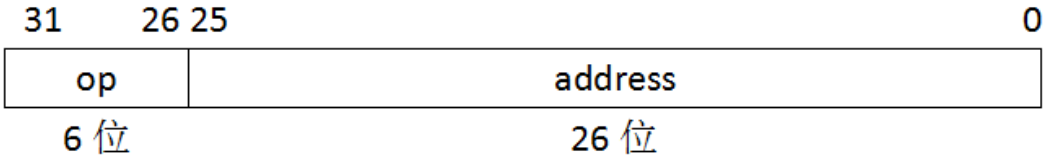
R 类型：



I 类型：



J 类型：



其中，

op：为操作码；

rs：为第 1 个源操作数寄存器，寄存器地址（编号）是 00000~11111，00~1F；

rt：为第 2 个源操作数寄存器，或目的操作数寄存器，寄存器地址（同上）；

rd：为目的操作数寄存器，寄存器地址（同上）；

sa：为位移量（shift amt），移位指令用于指定移多少位；

funct：为功能码，在寄存器类型指令中（R 类型）用来指定指令的功能；

immediate：为 16 位立即数，用作无符号的逻辑操作数、有符号的算术操作数、数据加载(Load)/数据保存(Store)指令的数据地址字节偏移量和分支指令中相对程序计数器(PC)的有符号偏移量；

address：为地址。

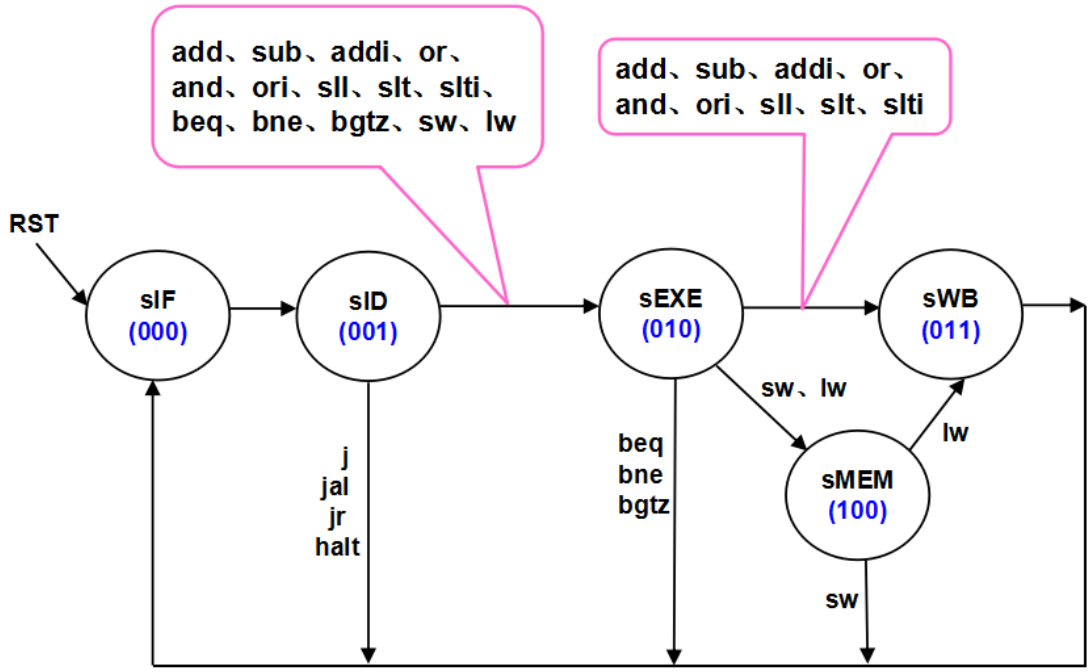


图 2 多周期 CPU 状态转移图

状态的转移有的是无条件的，例如从 sIF 状态转移到 sID 就是无条件的；有些是有条件的，例如 sEXE 状态之后不止一个状态，到底转向哪个状态由该指令功能，即指令操作码决定。每个状态代表一个时钟周期。

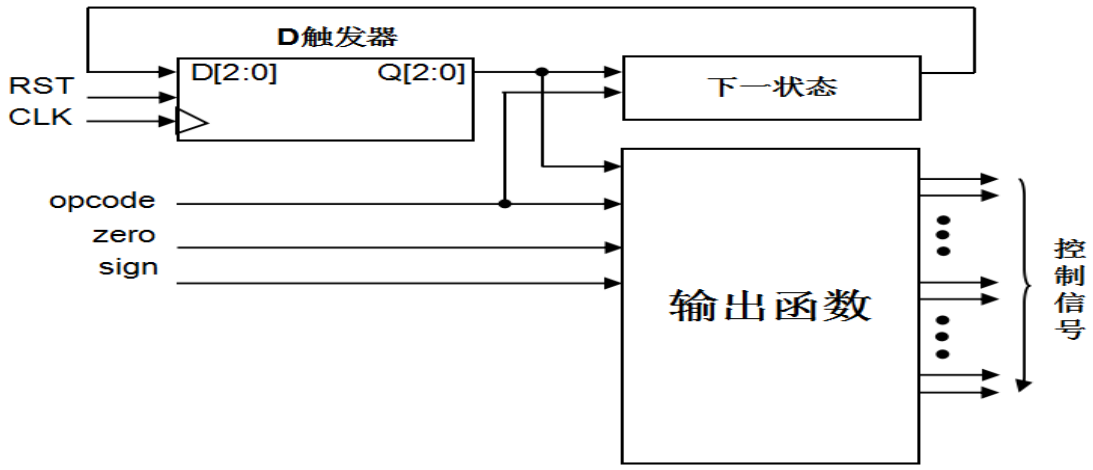


图 3 多周期 CPU 控制部件的原理结构图

图 3 是多周期 CPU 控制部件的电路结构，三个 D 触发器用于保存当前状态，是时序逻辑电路，RST 用于初始化状态“000“，另外两个部分都是组合逻辑电路，一个用于产生下一个阶段的状态，另一个用于产生每个阶段的控制信号。从图上可看出，下个状态取决于指令操作码和当前状态；而每个阶段的控制信号取决于指令操作码、当前状态和反映运算结果的状态 zero 标志和符号 sign 标志。

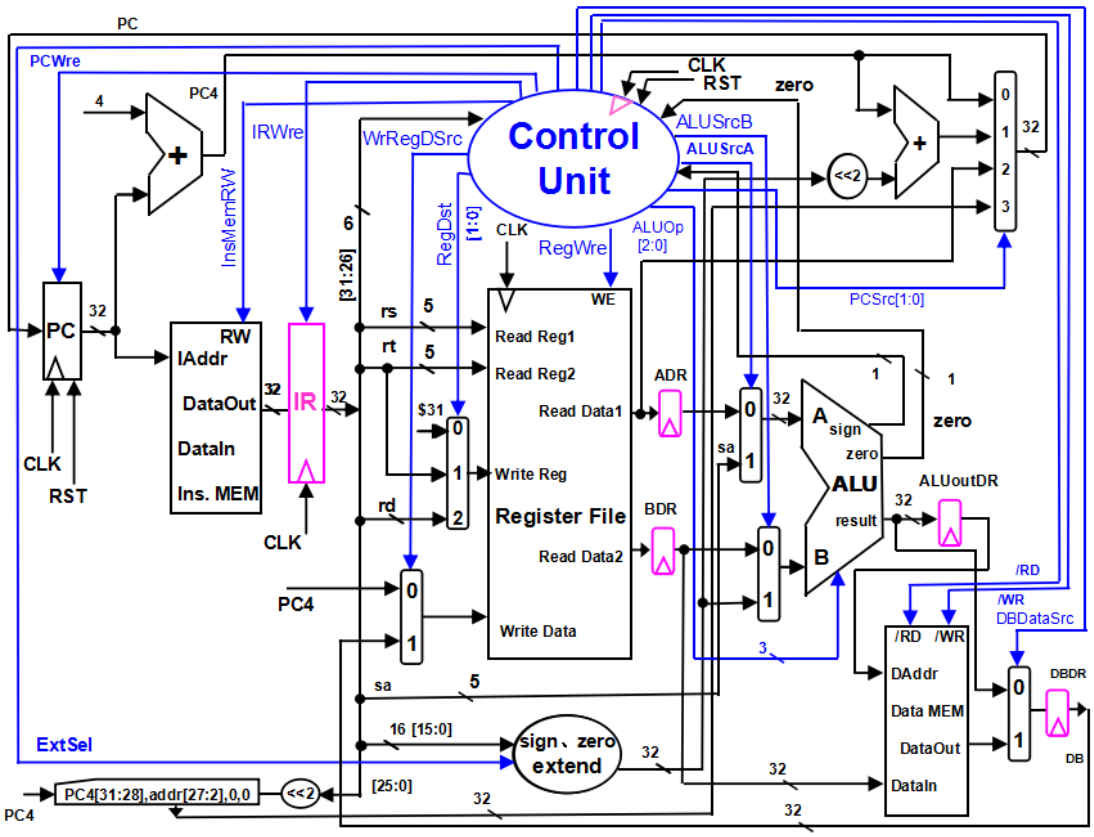


图 4　多周期 CPU 数据通路和控制线路图

图 4 是一个简单的基本上能够在多周期 CPU 上完成所要求设计的指令功能的数据通路和必要的控制线路图。其中指令和数据各存储在不同存储器中，即有指令存储器和数据存储器。访问存储器时，先给出内存地址，然后由读或写信号控制操作。对于寄存器组，给出寄存器地址（编号），读操作时，输出端就直接输出相应数据；而在写操作时，在 WE 使能信号为 1 时，在时钟边沿触发将数据写入寄存器。图中控制信号功能如表 1 所示，表 2 是 ALU 运算功能表。

特别提示，图上增加 IR 指令寄存器，目的是使指令代码保持稳定，pc 写使能控制信号 PCWre，是确保 pc 适时修改，原因都是和多周期工作的 CPU 有关。ADR、BDR、ALUoutDR、DBDR 四个寄存器**不需要写使能信号，**其作用是切分数据通路，将大组合逻辑切分为若干个小组合逻辑，大延迟变为多个分段小延迟。

表 1　控制信号作用

控制信号名	状态“0”	状态“1”
RST	对于 PC，初始化 PC 为程序首地址	对于 PC，PC 接收下一条指令地址
PCWre	PC 不更改，相关指令：halt，另外，除‘000’状态之外，其余状态慎改 PC 的值。	PC 更改，相关指令：除指令 halt 外，另外，在‘000’状态时，修改 PC 的值合适。
ALUSrcA	来自寄存器堆 data1 输出，相关指令：add、sub、addi、or、and、ori、beq、bne、bgtz、slt、slti、sw、lw	来自移位数 sa，同时，进行(zero-extend)sa，即 {{27{0}},sa}，相关指令：sll
ALUSrcB	来自寄存器堆 data2 输出，相关指	来自 sign 或 zero 扩展的立即数，相关

	令：add、sub、or、and、beq、bne、bgtz、slt、sll	指令：addi、ori、slti、lw、sw
DBDataSrc	来自 ALU 运算结果的输出，相关指令：add、sub、addi、or、and、ori、slt、slti、sll	来自数据存储器（Data MEM）的输出，相关指令：lw
RegWre	无写寄存器组寄存器，相关指令：beq、bne、bgtz、j、sw、jr、halt	寄存器组寄存器写使能，相关指令：add、sub、addi、or、and、ori、slt、slti、sll、lw、jal
WrRegDSrc	写入寄存器组寄存器的数据来自 pc+4(pc4)，相关指令：jal，写$31	写入寄存器组寄存器的数据来自 ALU 运算结果或存储器读出的数据，相关指令：add、addi、sub、or、and、ori、slt、slti、sll、lw
InsMemRW	写指令存储器	读指令存储器(Ins. Data)
/RD	读数据存储器，相关指令：lw	存储器输出高阻态
/WR	写数据存储器，相关指令：sw	无操作
IRWre	IR(指令寄存器)不更改	IR 寄存器写使能。向指令存储器发出读指令代码后，这个信号也接着发出，在时钟上升沿，IR 接收从指令存储器送来的指令代码。与每条指令都相关。
ExtSel	(zero-extend)**immediate**，相关指令：ori；	(sign-extend)**immediate**，相关指令：addi、lw、sw、beq、bne、bgtz；
PCSrc[1..0]	00：pc<－pc+4，相关指令：add、addi、sub、or、ori、and、slt、slti、sll、sw、lw、beq(zero=0)、bne(zero=1)、bgtz(sign=1，或 zero=1)； 01：pc<－pc+4+(sign-extend)**immediate**，相关指令：beq(zero=1)、bne(zero=0)、bgtz(sign=0，且 zero=0)； 10：pc<－rs，相关指令：jr； 11：pc<－{pc[31:28],addr[27:2],0,0}，相关指令：j、jal；	
RegDst[1..0]	写寄存器组寄存器的地址，来自： 00：0x1F($31)，相关指令：jal，用于保存返回地址（$31<-pc+4）； 01：rt 字段，相关指令：addi、ori、slti、lw； 10：rd 字段，相关指令：add、sub、or、and、slt、sll； 11：未用；	
ALUOp[2..0]	ALU 8 种运算功能选择(000-111)，看功能表	

相关部件及引脚说明：

Instruction Memory：指令存储器

Iaddr，指令地址输入端口

DataIn，存储器数据输入端口

DataOut，存储器数据输出端口

RW，指令存储器读写控制信号，为 0 写，为 1 读

Data Memory：数据存储器

Daddr，数据地址输入端口

DataIn，存储器数据输入端口

DataOut，存储器数据输出端口

/RD，数据存储器读控制信号，为 0 读

/WR，数据存储器写控制信号，为 0 写

Register File：寄存器组

Read Reg1，rs 寄存器地址输入端口

Read Reg2，rt 寄存器地址输入端口

Write Reg，将数据写入的寄存器，其地址输入端口（rt、rd）

Write Data，写入寄存器的数据输入端口

Read Data1，rs 寄存器数据输出端口

Read Data2，rt 寄存器数据输出端口

WE，写使能信号，为 1 时，在时钟边沿触发写入

IR：　**指令寄存器**，用于存放正在执行的指令代码

ALU：　算术**逻辑单元**

result，ALU 运算结果

zero，运算结果标志，结果为 0，则 zero=1；否则 zero=0

sign，运算结果标志，结果最高位为 0，则 sign=0，正数；否则，sign=1，负数

表 2 ALU 运算功能表

ALUOp[2..0]	功能	描述
000	Y = A + B	加
001	Y = A - B	减
010	Y= (A<B) ?1: 0	比较 A 与 B 不带符号
011	**if (A<B && (A[31] == B[31]))　Y = 1;** **else if (A[31] ==1 && B[31] == 0)　Y = 1;** **else　Y = 0;**	比较 A 与 B 带符号
100	Y = B<<A	B 左移 A 位
101	Y = A ∨ B	或
110	Y = A ∧ B	与
111	Y = A ⊕ B	异或

值得注意的问题，设计时，用模块化、层次化的思想方法设计，关于如何划分模块、如何整合成一个系统等等，是必须认真考虑的问题。

四、实验设备

PC 机一台，BASYS 3 实验板一块，Xilinx Vivado 开发软件一套。

五、其它要求事项

（1）电子文档必须按如下规范：

实验报告电子文档：ECOP-学号-XX.DOC，其中 XX：代表第几次实验，如 01、02...；

必须注意：“ECOP”与“学号”与“XX”用“-”连接，而不用“_”。

其它相关的设计文档：ECOP-学号-XX.RAR，全部打包在该文件中。

以上**两个文件独立存放**，但必须同时提交。如果不交这两个文档，本次实验没成绩。

（2）实验必须在规定时间内完成，尽量在正常上课时间内接受提问性实验检查，而且**每位同学都必须通过检查这一关**，这样本次实验才算完成。不在规定时间内完成的实验，扣分。

（3）实验报告必须按模板要求严格执行，不合规范、书写简单及随便表达文意等等，扣分。必须注意：**如果发现实验报告有抄的嫌疑，扣分是很重的**。

（4）注意：**没有通过检查的实验，如果只提交相关电子文档，该次实验不计入成绩**。

（5）若在非正常上课时间检查实验，（如果我在实验室）周一到周五，请在这个时间段：9:30-17:30，当然 11:30-14:00 是午餐与休息时间。