基于 FPGA 的五级流水线 CPU①

王绍坤

(北京理工大学 计算机学院, 北京 100081)

摘 要: 基于 FPGA 平台设计并实现了一种五级流水线 CPU. 它参考 MIPS 机将指令的执行过程进行抽象, 把指令分成取值、译码、执行、访存、写回五级流水处理. 首先设计系统级的结构, 决定 CPU 的结构和指令系统. 其次对整体结构进行分解, 确定模块与模块之间的信号连接, 采用 VHDL 实现 CPU. 最后通过 Debug-controller 调试软件对五级流水线 CPU 进行调试. 结果表明了所设计的流水线 CPU 的有效性.

关键词: VHDL; FPGA; 流水线 CPU

Five Stage Pipeline CPU Based on FPGA

WANG Shao-Kun

(School of Computer Science, Beijing Institute of Technology, Beijing 100081, China)

Abstract: A five stage pipeline CPU was designed and implemented on FPGA. Referring to MIPS machine and analyzing the process of each instruction, the process was divided into five stages which are IF, ID, EXE, MEM, and WB. The design of system-level structure was placed in the first position in order to determine the architecture and the instruction set. The next work was decomposing the integrated architecture and determining the signal connection between the module and the module. The CPU was implemented with VHDL. Finally, the five stage CPU was debugged by debugging software which is called Debug-controller. The result shows that the pipeline CPU is effective.

Key words: VHDL; FPGA; pipeline CPU

1 引言

现在集成电路设计进入 SOC(System On Chip)时代. SOC 一般以 CPU 为核心, 集成控制电路和存储器, 完成系统中信息处理的主要功能. 在性能和成本方面, SOC 和传统的板上系统具有无法比拟的优势. 而 CPU 作为 SOC 的核心部件, 其性能直接影响整个系统的性能. 因此, 如何设计与实现有效的 CPU 核已成为 SOC 的关键问题.

MIPS(Microprocessor without Interlocked Pipeline Stages) 是一种优秀的、开放的 RISC(Reduced Instruction Set Computer)CPU 体系结构, 在 SOC 应用领域得到广泛的应用. FPGA(Field-Programmable Gate Array),为现场可编程门阵列, 能够提供灵活设计的平台, 由于硬件描述语言的普及和集成度的大幅提高, 使人们能够自主的设计 CPU. FPGA 技术的迅速发展, 由最初应用到通信领域, 到目前, 已被广泛应用到信息产业的诸多领域, 比如: 航空, 医疗, 通讯等.

本文参考 MIPS 流水线 CPU 设计理念, 以 Altera 公司的 FPGA 为目标器件, 采用了自顶向下的层次模块化设计方法, 用 VHDL 语言描述了 16 位的流水线 CPU. 该 CPU 以《开放式实验 CPU 设计》一书中非流水线 CPU 为原型, 将指令的执行过程分割成取值, 译码, 执行, 访存, 写回五段, 并且将原来的非流水线体系结构改造成流水线体系结构. 首先设计系统级的结构, 决定 CPU 的结构和指令系统, 其次对模块进行分解, 确定模块与模块之间的信号连接, 再次进行电路级的管脚定义, 下载并烧片, 形成一个物理的 CPU, 最后通过外部调试程序运行汇编程序来对 16 位流水线 CPU 进行调试, 最终确定一个正确无误的流水线 CPU.

① 收稿时间:2014-07-04;收到修改稿时间:2014-08-12

2 流水线CPU工作原理及指令系统

2.1 流水线 CPU 工作原理

流水线是提高 CPU 运行效率的关键技术, 其核心思想是把多条指令的不同执行阶段重叠起来, 使得 CPU 能同时处理多条指令. 这些指令分处于不同的运行周期, 使用不同的物理器件. 在流水线 CPU 中, 每条指令的执行过程被分成若干个执行阶段. 只有当每一个指令执行阶段都完成之后, 一条指令才算执行完毕. 在每一个指令执行阶段中, 当一条指令在该阶段执行完后, 下一条指令将立即进入到该执行阶段来执行操作. 本论文描述的流水线 CPU 将每条指令划分为“取指”、“译码”、“执行”、“访存”、“写回”这五个执行步骤. 当流水线处于饱和状态时, 在 CPU 中, 将同时有五条指令在同时执行. 如下图 1 所示.

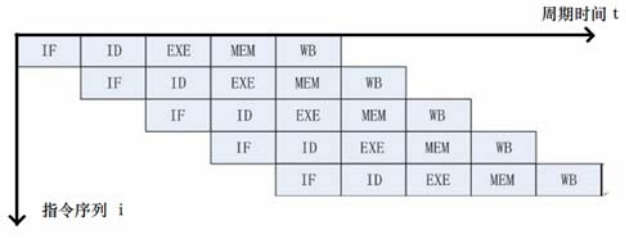

图 1 五级流水线 CPU 时空图

2.2 设计指令系统

指令系统设计的好坏影响整个系统各个方面的性能. 本论文直接采用《开放式实验 CPU 设计》一书中非流水线 CPU 的指令系统. 该指令系统包含了 5 种指令类型, 共 15 条指令, 如图 2 所示.

类型	汇编语句	功能描述	指令格式
算数指令	ADD DR,SR	DR <- DR+SR	0000 DR SR 0000 0111
	INC DR	DR <- DR+1	0001 DR SR 0000 0111
	SUB DR,SR	DR <- DR-SR	0010 DR SR 0000 0111
	DEC DR	DR <- DR-1	0011 DR SR 0000 0111
逻辑指令	AND DR,SR	DR <- DR and SR	0100 DR SR 0000 0011
	OR DR,SR	DR <- DR or SR	0101 DR SR 0000 0011
	NOT DR	DR <- not DR	0110 DR SR 0000 0011
跳转指令	JMP ADR	PC <- ADR	1000 0000 0000 0000
			ADR
	JNC ADR	C=0 PC <- ADR	1001 0000 ADR-PC-1
		C=1 PC<- PC+1	
	JNZ ADR	Z=0 PC <- ADR	1010 0000 ADR-PC-1
		Z=1 PC<- PC+1	
数据传送指令	MOV DR,SR	DR <- SR	0111 DR SR 0000 0001
	MVDR DR,DATA	DR <- DATA	1100 0000 0000 0000
			DATA
	LDR DR,SR	DR <- [SR]	1101 DR SR 0000 0001
	STR SR,DR	[DR] <- SR	1110 DR SR 0000 0000
空指令	NOP	Do nothing	0000 0000 0000 0000

图 2 五级流水线 CPU 指令系统

DR 和 SR 分别表示寄存器的编号. DR <-DR+SR 表示将编号为 DR 和 SR 寄存器数据相加, 结果写入到编号为 DR 的寄存器中. JMP 和 MVDR 为双字指令, ADR 和 DATA 均表示 16 位数据. C,Z 分别代表运算器进位标志和清零标志. DR <- [SR]表示编号为 SR 寄存器中的内容作为内存的地址, 该地址里面的数据赋值给编号为 DR 的寄存器.

3 流水线CPU面临的问题和解决方案

3.1 结构相关

由于我们的设计中不包含缓存, 且指令和数据存放在同一个存储器中, 当执行访存指令时就会和取指阶段冲突, 即会有取指和访存的冲突, 这叫做结构相关. 当冲突发生时, 必须先执行“访存”, 将“取指”延后一个时钟周期, 这样才能保证指令的正确执行. 在执行阶段进行冲突检测, 在检测到处于执行阶段的指令是访存指令时, 则将 PC 值保持不变, 并向指令寄存器 IR 中写入空指令 NOP.

3.2 数据相关

数据相关是指在执行本条指令的过程中, 如果用到的操作数是前面指令的执行结果, 则必须等待前面的指令执行完成, 并把结果写回寄存器或主存之后, 本条指令才能继续执行.

本论文采用设置专用数据通路(即旁路技术)来解决数据相关问题. 但若前一条指令是 LOAD 指令, 而后一条指令和其数据相关, 要用到 LOAD 给出的数据, 这时 LOAD 指令尚未给出数据. 针对这种情况, 本论文采用最简单的方式在 LOAD 指令后面增加一条空指令 NOP, 即增加冗余的方法来解决.

根据有数据相关的指令之间的关系分成三种情况.

(1) 相邻指令数据相关. 举例如下:

INC R0

INC R0

在此情形下, 将上一指令的 ALU 输出暂存后直接送回, 作为 ALU 多路选择器的入口之一.

(2) 中间隔 1 条指令的两指令数据相关, 举例如下:

INC R0

INC R1

INC R0

在此情形下, 将第 1 条指令的写回数据送回作为 ALU 多路选择器的入口之一供第 3 条指令使用.

(3) 中间隔 2 条指令的两指令数据相关. 举例如下:

INC　R0

INC　R1

INC　R2

INC　R0

考虑到读写寄存器的时间较短, 因此将写寄存器的时机改在时钟下降沿, 这样第一条指令和第四条指令之间就没有了数据相关. 分析: 当第一条指令处于“译码阶段”, 第四条指令处于“写回”阶段, 当在时钟的下降沿改变寄存器的值时, 所以在时钟的后半部分寄存器的值就得到最新的值了, 所以第一条指令读出寄存器的值仍然是最新的值, 所以这样 1、4 指令就没有数据相关了.

3.3 控制相关

除 JMP 外, JNC,JNZ 等条件跳转需要根据当前状态位来决定是否跳转, 而当前状态位是由前面最近的会影响状态位的指令(如算术指令)决定. 本论文的做法是在取指阶段遇到 JNC,JNZ 指令时就往流水线上插入 NOP 指令.

当 JNC,JNZ 指令处于译码阶段的前半周期时, 处于执行阶段的指令就能产生最新的 C,Z 值, 将这些最新的 C,Z 值直接送往处于译码阶段的 JNC,JNZ 指令, 则 JNC,JNZ 指令就能判断是否进行跳转, 在时钟的下降沿改变 PC 的值就能知道下一条指令的地址是跳转地址还是 PC 加 1.

本论文解决控制相关需要做三项工作: ①每当流水线中进来一条新的指令, 则判断该指令是否是 JNC,JNZ 指令, 若是则保持 PC 值不变, 往 IR 寄存器中插入 NOP 指令. ②在时钟的前半周期,获得最新的 C,Z 值, 直接送到译码阶段判断是否跳转成功. ③选择 PC 更新的时机, 将 PC 的更新放在时钟的下降沿.

3.4 双字指令的处理

本指令系统中 MVDR 指令和 JMP 指令是双字指令, 假如读取的指令是双字指令时则必须两次读内存把该条指令读取出来, 并且在第二次读内存之后得向流水线里插入 NOP 指令.

4 流水线CPU的各个模块及总体逻辑结构

4.1 流水线 CPU 的各个功能模块

整个流水线 CPU 由以下几个部分组成:

(1) 取指模块

取指模块要做的工作是给出内存地址, 读取指令并送入到指令寄存器. 当处于译码阶段的指令为 JNC、JNZ 指令时, 假如跳转成功则在时钟的下降沿根据译码阶段给出的地址更新 PC. 假如处于译码阶段的指令为 JMP 指令, 则将内存中读出的数据直接赋值给 PC. 当检测到控制相关和结构相关则将 PC 的值保持不变且往流水线中插入 NOP 指令.

(2) 译码模块

译码模块根据处于译码阶段的指令寄存器的值产生流水线 CPU 所需要的各种控制信号和其它信号. 产生条件转移指令(JNC 和 JNZ)所需要的转移标志 flag 和跳转地址 addr.

(3) 执行模块

执行模块完成下列任务: ①执行 8 种算数逻辑运算: 加、加 1, 减, 减一, 与, 或, 非和数据传送, 并产生进位标志 c_out 和 0 标志位 z_out. ②根据旁路通路给出的数据相关的控制信号正确选择操作数. ③产生访问存储器的地址信号 alu_addr.

(4) 访存模块

访存模块要做的工作是选择地址线的数据来源和数据线的流向. 访存和取指在功能上是独立的, 但 CPU 对外只有一条地址线和数据线的事实决定了访存和取指是相互联系的. 当执行 LOAD/STORE 指令时, 地址线由 ALU 送入“访存段”的值提供; 取指时, 则由 PC 提供. 当写内存时, CPU 内部数据送数据线; 当需要读内存时, CPU 往数据线送高阻.

(5) 写回模块

当处于写回模块的指令需要写目的寄存器时提供写入寄存器组的数据以及写入寄存器的编号以便将数据写回到寄存器组中.

(6) 寄存器组模块

在本设计中寄存器组中总共有四个寄存器. 只有具有写目的寄存器功能的指令在写回阶段时才写目的寄存器. 有些指令, 如 JMP 指令不会改变通用寄存器的值. 通用寄存器组的两个读出端口, 一个是目的寄存器读出端口, 一个是源寄存器读出端口, 从这两个端口读出的内容供执行部件使用. 通用寄存器组内还

有两个标志位: z_out 和 c_out, 在时钟的下降沿将执行部件的 c 和 Z 标志位写入到 z_out 和 c_out.

(7) Forwarding 模块

Forwarding 模块是用来检测数据相关. 即检测处于译码阶段的指令是否和处于执行及访存阶段的指令有数据相关.

(8) 控制相关检测模块

该功能模块检测处于取指阶段的指令是否为 JNC、JNZ、LDR、STR 指令或者是否出现了结构相关即处于执行阶段的指令是访存指令 LDR 或 STR 指令, 假如是的话则将 PC 的值暂停, 并且往流水线中插入 NOP 指令.

4.2 流水线 CPU 总体逻辑结构

流水线 CPU 总体逻辑结构如图 3 所示.

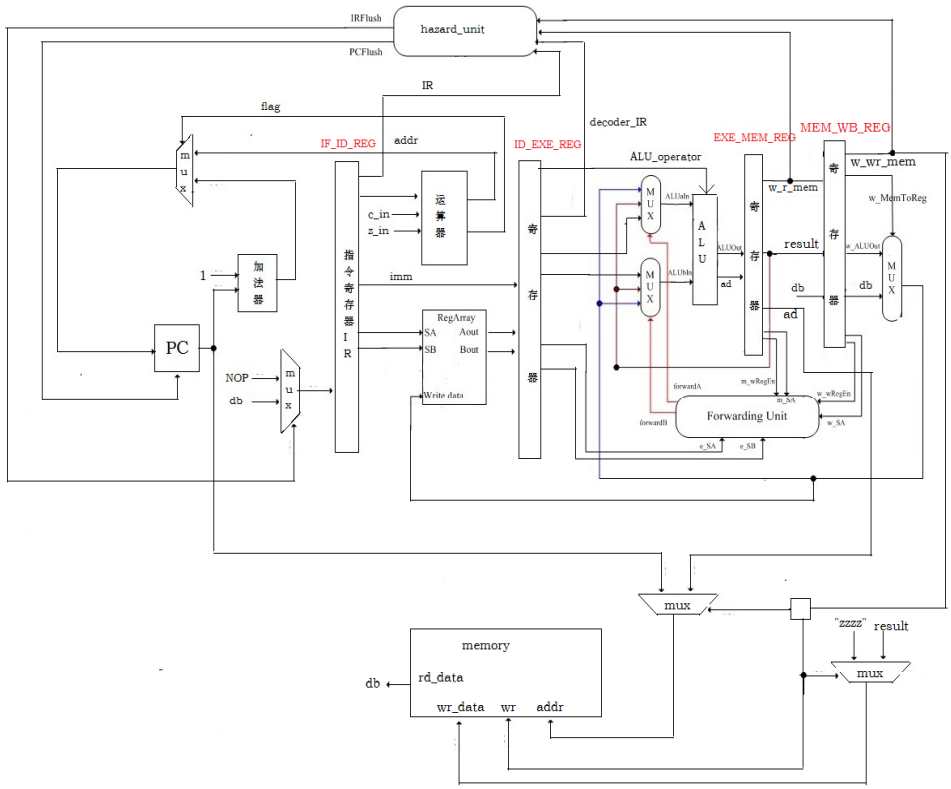

图 3 流水线 CPU 整体逻辑结构图

hazard_unit 模块用于检测控制相关. 当检测到控制相关发生时, PCFlush 输出为 1, 并且暂停 PC 值. 计算条件跳转指令 JNC,JNZ 的目标地址放到译码模块, 当跳转成功时, flag=1,将计算的目标地址赋值给 PC. 若跳转不成功或者处于译码阶段的指令不是条件跳转指令, 则 flag=0,将 PC+1 赋值给 PC.

Forwarding_unit 模块用于检测数据相关, 输出控制相关信号 forwardA 和 forwardB. 该信号分别用于充当执行模块中两个三路选择器的控制信号, 该三路选择器的输入分别是寄存器数据、上一条指令 ALU 的输出结果 result 和来自访存模块的数据.

memory 模块用于访问外部内存单元, 提供地址信号 addr, 读写控制信号 wr,写内存数据 wr_data, 以及读内存数据 db. 当 LDR, STR 指令处于访存阶段时, addr 由访存模块的 ad 提供, 在其他情况下, addr 由 PC 提供. 当 STR 访存指令处于写回阶段时, wr_data 由访存模块 result 提供, 否则为高阻状态.

5 测试结果及分析

采用 Debugcontroller 软件调试流水线 CPU. 该软件功能有两个: (1)将用户按照自己定义的指令集编写的汇编形式调试程序转换成机器代码, 并将它装入实验平台上的存储器中, 机器代码的格式根据用户定义的指令集而定. (2)调试用户设计的 CPU 及用户编写的调试程序. 该软件支持单步调试和断点调试.

两个测试文件 T1.txt 和 T2.txt 分别检测数据相关

和结构相关、控制相关. 这两个测试文件如图 4 所示.

```
main: MVRD    R2, 0x5555
      MVRD    R3, 0xffff
      ADD     R3, R2
      SUB     R3, R2
      INC     R3
      DEC     R3
      OR      R3, R2
      AND     R3, R2
      NOT     R3
      MOV     R2, R3
      NOP
      NOP
      NOP
      NOP
      NOP
```

T1.txt

```
main: MVRD    R0, 0x100
      MVRD    R1, 0x200
      MVRD    R2, 0Xf
      MVRD    R3, 0x3500
T1:   INC     R3
      INC     R0
      DEC     R2
      JNC     T1
T2:   MVRD    R0, 0x100
      MVRD    R2, 0xf
T3:   LDR     R3, R0
      INC     R0
      INC     R1
      DEC     R2
      JNC     T3
T4:   NOP
      NOP
      JMP     main
```

T2.txt

图 4　测试文件

测试文件 T1.txt 用于检测数据相关. R2 和 R3 都是 16 位寄存器. 当该汇编程序执行到最后时, R2 寄存器的内容变成 0xAAAA, R3 寄存器的内容变成 0xAAAA. 测试文件 T2.txt 用于检测控制相关和结构相关. 其程序的具体分析如图 5 所示.

```
main: MVRD    R0, 0x100      // R0=0x100
      MVRD    R1, 0x200      // R1=0x200
      MVRD    R2, 0Xf        // R2=0Xf
      MVRD    R3, 0x3500     // R3=0x3500
T1:   INC     R3             //R3++
      INC     R0             //R0++
      DEC     R2             //R2--
      JNC     T1             //当 R2 非零跳转，该段程序循环十六次
//以上程序执行完成之后，R3=0x3510，R2=0x0000，R1=0x200，R0=0x110
T2:   MVRD    R0, 0x100      //R0=0x100
      MVRD    R2, 0xf        //R2=0xf
T3:   LDR     R3, R0         //内存地址为 R0 中的数据赋值给 R3 寄存器
      INC     R0             //R0++
      INC     R1             //R1++
      DEC     R2             //R2--
      JNC     T3             //当 R2 非零跳转，该段程序循环十六次
T4:   NOP
      NOP
      JMP     main
//程序执行完毕之后 R0=0x110,R1=0x210,R2=0xffff,R3=内存地址为 0x10f
的内容
```

图 5　T2.txt 测试文件分析

5.1 数据相关检测

测试程序 T1.txt 运行的中间结果如图 6 所示.

分析: Debugcontroller 软件包括寄存器窗口, 内存窗口, 能够实时检测 FPGA-CPU 内部的寄存器以及存储器窗口, 实时显示内存中的内容. Reg 00~Reg 03 表示寄存器组的 R0~R3 的值, Reg 04~Reg 07 代表译码, 执行, 访存和写回阶段的指令寄存器的值. Reg 09 表示 ALU 执行部件执行后的结果.

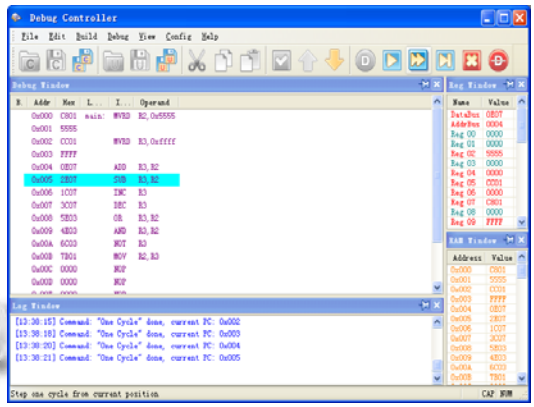

图 6　当流水线 CPU 要处理 SUB 指令时的结果

如图 6 所示, 当执行到 SUB R3, R2 指令时, MVRD R2,0x5555 正好执行完“写回”完毕, 将结果写回到寄存器 R2 中, 此时 R2 的值变为 0x5555. 所以该流水线 CPU 实现了双字指令的处理. Reg07 表示处于写回阶段的指令. Reg07 为 0XC801 和内存中地址为 0X0000 中的内容相一致. Reg06 为 0x0000 这是因为当处理双字指令是得需要把内存中的数据读出来之后往流水线中插入 NOP 指令. Reg05 的数据为 0xcc01 和内存中地址 0x0002 中的内容相一致. Reg04 中的内同为 0x0000 也是因为当处理双字指令是得需要把内存中的数据读出来之后往流水线中插入 NOP 指令. Reg09 代表处于执行阶段的计算结果, 此时 MVRD R3,0xFFFF 处于执行阶段, 结果是 0xFFFF.

测试程序 T1.txt 运行的最终结果如图 7 所示

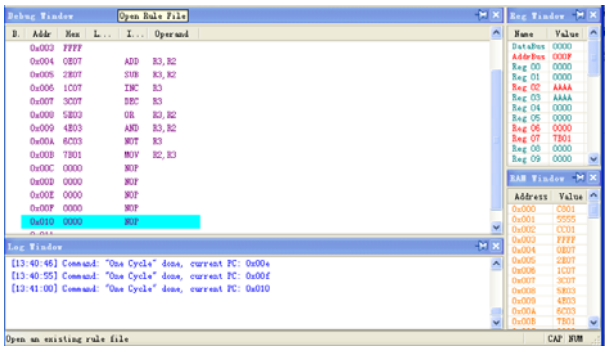

图 7　T1.txt 测试文件执行完的结果

分析: 此时 MOV R2, R3 指令写回阶段执行完毕的结果, 此时 R2=R3=0xAAAA 和预期的结果一致, 说

明流水线 CPU 消除了数据相关, 正确执行了 T1.txt 测试文件.

5.2 结构相关和控制相关检测

测试程序 T2.txt 在 CPU 上运行的中间结果如图 8 所示.

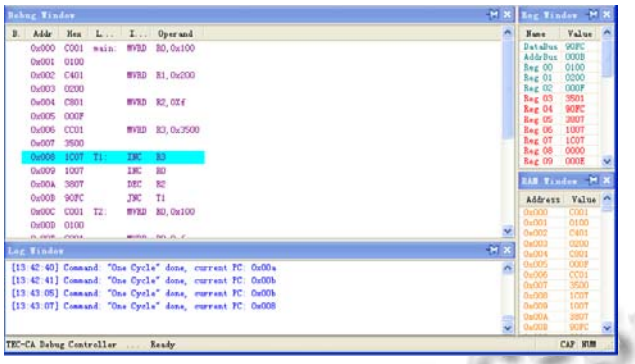

图 8 当流水线 CPU 执行 JNC 指令时的结果

结果分析: PC 值从 0x000b 跳转成 0x0008 说明正确执行了条件转转 JNC 指令.

当测试文件 T2.txt 在 CPU 上执行完 LDR 指令的结果如图 9 所示.

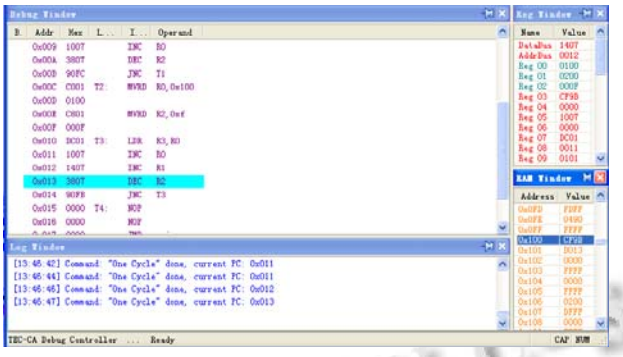

图 9 流水线 CPU 执行完 LDR 指令时的结果

结果分析: LDR R3, R0 指令执行完毕. 此时 R0=0x0100, 且内存中的 0x0100 中的内容写入到 R3 中. 实现了 LOAD 指令.

当测试文件 T2.txt 在 CPU 上运行执行完最后一条指令 JMP 指令的结果如图 10 所示.

结果分析: JMP 跳转指令成功. PC=0x0018 之后就直接跳转到 0x0000, 跳转成功, 正确执行了 JMP 指令.

执行结果 R0=0x110, R1=0x210, R2=0xFFFF, R3=0xFFFF, 此时内存地址为 0x10f 的内容也为 0xFFFF 和预期的运算结果完全一致, 说明了该流水线 CPU 正确执行了 T2.txt 测试程序. 由以上分析可知, 所设计的流水线 CPU 有效地执行了调试程序.

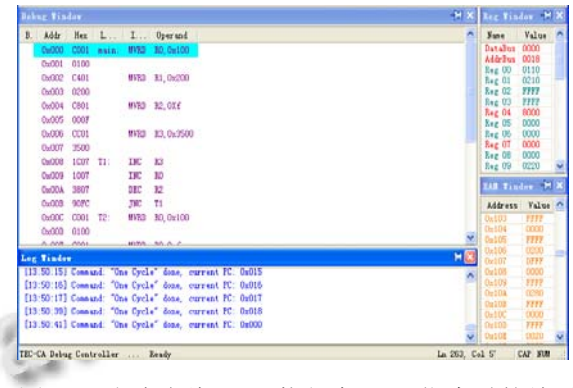

图 10 当流水线 CPU 执行完 JMP 指令时的结果

6 结语

流水线是现代 CPU 中普遍采用的一种技术, 它只需要增加很少的硬件就能使 CPU 的速度提高很多. 本论文的设计过程是首先进行系统级的结构, 决定 CPU 的结构和指令系统, 其次对总体结构进行分解, 确定模块与模块之间的信号连接, 再次进行电路级的针脚定义, 下载并烧片, 形成一个物理的 CPU, 最后通过外部调试程序运行汇编程序来对 16 位流水线 CPU 进行调试, 最终确定一个正确无误的流水线 CPU.

本文作者的创新点: 根据五级流水线的设计理念, 将《开放式实验 CPU 设计》一书中非流水线 CPU 改造成具有 MIPS 风格和理念流水线 CPU, 即提出了一种将特定非流水线 CPU 改造成流水线 CPU 的解决方案.

参考文献

1 汤志忠,杨春武.开放式实验 CPU 设计.北京:清华大学出版社,2007.
2 田俊峰.计算机系统结构.北京:机械工业出版社,2006.
3 侯伯亨,刘凯,顾新.VHDL 硬件描述语言与数字逻辑电路设计(第 3 版).西安:西安电子科技大学出版社,2009.
4 郑亚民,董晓舟.可编程逻辑器件开发软件 Quartus II.北京:国防工业出版社,2001.
5 白中英.计算机组成原理.北京:科学出版社,2001.