

自主设计精简指令集的流水线 CPU

袁 婷, 刘怡俊

(广东工业大学 计算机学院, 广东 广州 510006)

摘 要: 介绍了基于 FPGA 平台, 设计 16 位精简指令集流水线 CPU. 该 CPU 参考 MIPS 架构设计精简指令集, 通过分析指令处理过程实现五级流水线结构, 结合“预测技术”和数据前推方法解决流水线相关问题. 为了支持 CPU 软件架构, 设计指令集的汇编编译器. 在 Modelsim 平台运行测试程序, 给出仿真综合结果. 通过试验结果对比表明, 所设计的 CPU 处理过程所需时钟周期大大减少.

关键词: CPU; RISC; 流水线; 相关性; 汇编器

中图分类号: TP332

文献标识码: A

文章编号: 1000-7180(2015)02-0124-05

The Independent Design of RISC Pipeline CPU

YUAN Ting, LIU Yi-jun

(School of Computer, Guangdong University of Technology, Guangzhou 510006 China)

Abstract: This paper based on FPGA platform, designing a 16 bit pipelined RISC CPU core. With reference to the MIPS instruction set, CPU completing the design of RISC instruction set, realizing five stage pipelined structure by analyzing instruction processing, combined with the "prediction" and data forwarding method to solve the related problems of pipeline. To support the CPU software architecture, we design a compiler for the instruction set. In the Modelsim platform running test program, the simulation results are given out. By comparing the experiment results, the clock cycles required is greatly reduced.

Key words: CPU; RISC; Pipelining; Dependency; Assembler

1 引言

随着微电子技术和计算机技术的不断发展, 将整个电子应用系统集成在单一的芯片上, 构成功能强大的完整的片上系统 (SoC), 已成为现代电子系统设计的趋势. 而作为 SoC 核心的 CPU 核, 其设计和实现尤为重要.

MIPS 是一种典型的指令集标准, 指令集开放、指令格式规整、易于流水线设计, 在嵌入式领域得到广泛应用. 为提高 CPU 的效率, 现代微处理器对于指令处理过程广泛采用流水线设计, 流水线结构提高了 CPU 的处理速度, 同时带来了相关性问题. 目前对于流水线相关性问题, 大多使用的方法是数据

前推技术和“延时”法, 当发生控制相关性问题时, 则至少需要暂停一个取指周期.

本文构建嵌入式 RISC 的体系结构. 参考 MIPS 设计精简指令集, 基于指令集完成汇编器的设计实现. 对每条指令进行分析, 设计五级流水线结构. 对于流水线引起的相关性问题, 采用数据旁路技术解决数据相关; 采用硬件“预测法”^[1] 以及“提前判断”解决控制相关, 提前对条件进行判断以及计算跳转的地址, 避免了暂停取指周期. 该 CPU 与传统方法设计的 CPU 相比, 处理速度得到较大提升.

2 体系结构

2.1 指令集^[2]

本文参考 MIPS 指令集的设计原则, 所设计的

收稿日期: 2014-04-01; 修回日期: 2014-05-26

基金项目: 国家自然科学基金项目“多核芯片异步片上网络的微电路和建模研究”(61106019); 广州市 2013 年科技重大专项项目“广州市集成电路设计 EDA 公共服务平台”(穗科信字[2013]164 号)

指令集长度为 16 位,指令格式主要有三种,分别是寄存器 R 类型、立即数 I 类型、跳转 J 类型.实现的指令集按照功能分为 5 种类型,包括算术运算类指令、逻辑运算类指令、条件跳转和无条件跳转类指令、数据传送类指令和特殊指令.特殊指令是一个停

机指令,使 CPU 暂停取指.该指令集共有 22 条指令,如表 1 所示,所有指令的运算都要通过寄存器,当需要和内存交换数据时,通过访存指令完成内存和寄存器的数据交换.指令寻址方式只有立即数寻址、寄存器寻址和寄存器相对寻址.

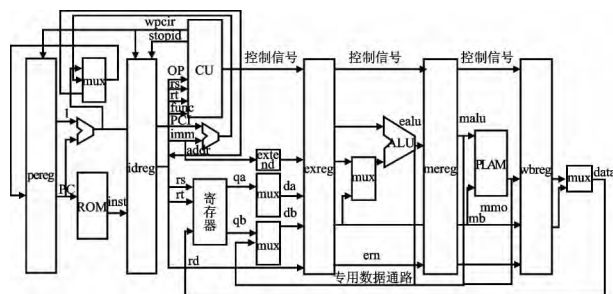
表 1 指令集

类型	指令	格式(位数)					指令描述
		4	3	3	3	3	
R 型	ADD rd, rs, rt	0000	rs	rd	rt	000	$R[rd] = R[rs] + R[rt]$
	SUB rd, rs, rt	0000	rs	rd	rt	001	$R[rd] = R[rs] - R[rt]$
	AND rd, rs, rt	0000	rs	rd	rt	010	$R[rd] = R[rs] \& R[rt]$
	OR rd, rs, rt	0000	rs	rd	rt	011	$R[rd] = R[rs] R[rt]$
	NOT rd, rs	0000	rs	rd	XXX	100	$R[rd] = \sim R[rs]$
	SHL rd, rs, sa	0000	rs	rd	sa	101	$R[rd] = R[rs] \ll sa$
	SHR rd, rs, sa	0000	rs	rd	sa	110	$R[rd] = R[rs] \gg sa$
	SAR rd, rs, sa	0000	rs	rd	sa	111	$R[rd] = R[rs] \gg sa$
	NOP	0000	000	000	000	000	空指令
	MOV rd, rs	1000	rs	rd	XXX	XXX	$R[rd] \leftarrow R[rs]$
I 型	ADDI rd, rs, X	0001	Rs	rd	立即数 X		$R[rd] = R[rs] + \text{sign_extend}(X)$
	SUBI rd, rs, X	0010	Rs	rd	立即数 X		$R[rd] = R[rs] - \text{sign_extend}(X)$
	ANDI rd, rs, X	0011	rs	rd	立即数 X		$R[rd] = R[rs] \& \text{un_extend}(X)$
	ORI rd, rs, X	0100	rs	rd	立即数 X		$R[rd] = R[rs] \text{un_extend}(X)$
	NOTI rd, X	0101	XXX	rd	立即数 X		$R[rd] = \sim \text{un_extend}(X)$
	LDM rd, B(rs)	0110	rs	rd	偏移地址 B		$R[rd] \leftarrow \text{Mem}[rs + \text{sign_extend}(B)]$
	STM rd, B(rs)	0111	rs	rd	偏移地址 B		$R[rd] \rightarrow \text{Mem}[rs + \text{sign_extend}(B)]$
	LDMI rd, X	1001	XXX	rd	立即数 X		$R[rd] = \text{sign_extend}(X)$
	BGZ rs, A	1010	rs	立即数 A			If $rs > 0$, $PC = PC + \text{sign_extend}(A)$
	BEZ rs, A	1011	rs	立即数 A			if $rs = 0$, $PC = PC + \text{sign_extend}(A)$
J 型	J A	1100	立即数 A				$PC = \text{un_extend}(A)$
	HLT	1111					停机

2.2 CPU 系统结构

本文将指令执行过程划分为五个阶段,分别是取指 IF、译码 ID、执行 EXE、访存 MEM、写回 WB. CPU 系统结构如图 1 所示,采用标准的哈弗结构,主要由 IF 级流水线 PC 寄存器 pcreg 及其组合电路、ID 级流水线寄存器 idreg 及其组合电路、EXE 级流水线寄存器 exreg 及其组合电路、MEM 级流水线寄存器 mereg 及其组合电路、WB 级流水线寄存器 wbreg 及其组合电路构成.

IF 级 PC 寄存器 pcreg 对下一条指令的地址进行存储,以便下一个周期根据地址进行正确地取指.组合电路包括 ROM 存储器模块、PC 加 1 模块、地址选择模块.

图 1 CPU 系统结构^[3]

ID 级流水线寄存器 idreg 存储上一级从 ROM 中取出的指令 ins 以及 PC1(本条指令地址加 1). 组合电路主要包括寄存器堆、数据扩展模块、控制器 (CU)、四选一选择器等等.

EXE 级流水线寄存器 *exreg* 存储 ID 级输出的扩展立即数 *X*、目的寄存器号 *rd*、选择器的输出结果以及控制器输出的控制信号等。组合电路主要包括 3 选 1 选择器, ALU 模块。

MEM 级流水线寄存器 *merreg* 存储 EXE 级输出的 ALU 结果、送存到 RAM 的数据、目的寄存器号以及 MEM 级的控制信号等。组合电路主要是 RAM 存储器模块。

WB 级流水线寄存器 *wbreg* 存储 ALU 的输出结果、RAM 存储器取出的数据、目的寄存器号以及 WB 级的控制信号。组合电路主要是二选一选择器, 从 ALU 计算结果和存储器取出的数据中选择一个作为 WB 级的输出。

3 流水线设计

3.1 相关性问题

根据指令的处理过程, 把指令处理过程进行抽象, 在空间上进行并行化^[1], 即实现流水线。流水线是提高 CPU 处理速度的关键技术, 而流水线的并行处理会产生指令的相关性问题, 本节主要讨论数据相关和控制相关问题。

数据相关, 即一条指令的执行依赖其上一条指令的运算输出结果, 那么这两条指令将会出现数据相关, 进而引起流水线的断流^[4]。针对数据相关问题, 采取数据前推技术解决^[5], 增加到 ID 级的旁路通路。增加 EXE 阶段到 ID 级的专用数据通路, 传输 ALU 的运算结果到 ID 级选择器; 增加 MEM 到 ID 级的数据通道, 传输寄存器储存的 ALU 输出结果到 ID 级选择器; 增加 MEM 到 ID 级的数据通道, 传输 ROM 读出的数据到 ID 级选择器。由于 *ldm* 指令从数据存储器读出的数据只能从 MEM 级前推到 ID 级, 这意味着 *ldm* 的后续指令如果与 *ldm* 指令发生数据相关问题, 需要流水线暂停一个周期^[4]。这里用控制信号 *wpcir* 作为流水线 PC 寄存器和 IF 与 ID 级之间的流水线寄存器的使能信号, 执行 *ldm* 指令时, *wpcir* 置为 0。

控制相关, 即执行到分支跳转指令时, 条件分支的条件还未判断或无条件分支的地址还没有计算出来, 此时与下一条指令发生控制相关。本设计中, CPU 发生控制相关时, 根据预测的下一条指令进行取指, 不用暂停取指。在译码阶段, 对条件分支指令进行条件判断, 计算出转移目标地址。判断条件满足, 置控制信号 *stopid* 为 0, 下一个时钟周期废弃预测的指令, 用空指令 NOP 代替预测的指令, 并根据

跳转地址取指。如下赋值语句, 由 *pcsource* 控制信号作为下一条地址的选择信号。

```
pcsource[1] = i_j;
pcsource[0] = i_j | i_bgz & rsgre | i_bez &
rsequ;
```

3.2 流水线各阶段

取指阶段, 根据程序计数器中的地址从 RAM 中取得指令; 完成 PC 自增 1 的计算, 同时根据控制信号 *pcsource* 从 PC 自增 1、条件分支地址、跳转地址中选择下一条指令的地址。

译码阶段, 对 16 位的指令进行译码; 从寄存器堆中读取 *rs* 和 *rt* 寄存器中数据; 若是 I 型或 J 型指令, 需要对立即数进行有符号或无符号扩展; 判断是否为分支跳转指令, 处理控制相关问题; 对比其前续两条指令的操作数 2 (目的寄存器) 判断是否发生数据相关, 若发生数据相关, 控制器产生暂停取指的信号 *wpcir*。控制器产生各个阶段的控制信号, 协调各个阶段的执行, 解决数据相关和控制相关性问题。

ID 级数据选择信号 *fwda* 和 *fwdb*, 从寄存器的数据、EXE 级 ALU 的输出结果、MEM 级 ALU 的输出结果、MEM 级从 RAM 存储器中读出的数据中选择一个正确输出。

sext: 在 ID 级对 I 型指令的后六位进行有符号或无符号扩展, 其中 *ADDI*、*SUBI*、*LDMI* 需要有符号扩展。

shift: 移位数据选择信号, 在 EXE 级, 选择 ALU b 使用移位数据 *sa*。

aluc[3:0]: ALU 操作控制信号, 为 X000 时进行加法运算; 为 X100 时进行减法运算; 为 X001 时进行与运算; 为 X101 时进行或运算; 为 X010 时进行取反操作; 为 1110 时, 执行 *ldm* 和 *stm* 指令的运算; 为 X011 时, 进行左移操作; 为 X111 时进行右移操作; 为 0110 时, 执行 *ldmi* 指令的操作。

mreg: 存储器数据写入寄存器信号, 在 WB 阶段选择存储器数据作为存入寄存器的数据。

wreg: 写寄存器信号, 在 WB 阶段控制寄存器的写入。

aluimm: 选择立即数作为 ALU b 的输入。

wmem: 写存储器信号, 在 MEM 阶段将外部数据写入到数据存储器中。

stopid: 条件跳转信号, ID 级在时钟上升沿时用空指令代替 IF 级的指令。

执行阶段, 从 *rt* 寄存器数据、移位数据 *sa*、立即数 *X* 中选择一个作为 ALU b 的输入, ALU a 的输

入是 rs 寄存器的数据,ALU 根据控制信号做相应的运算;访存阶段,对存储器进行读写操作;写回阶段,将 ALU 的计算结果或者存储器读出的数据写回到目的寄存器。

4 汇编器的设计实现

汇编器是处理器编译工具链中非常重要的一个环节,根据指令集编写的源代码是汇编语言,CPU 是无法识别的,必须要把汇编语言转换成机器可以识别的 0/1 目标代码。本文针对上述自主设计的指令集,设计了软件编程的支持架构,用 C 语言实现一个汇编器,包括的主要函数模块有主控模块、语法分析函数、目标代码生成函数等。汇编器处理过程如图 2 所示。

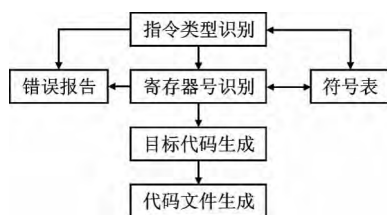


图 2 汇编器处理过程^[6]

汇编器逐行读取汇编代码文件,将指令的操作码助记符号(ADD、SUB 等)存入数组,利用 strcmp()函数与符号表对比判断指令类型;调用目标代码生成函数,生成操作码的目标代码,即 inst[15:12]和 inst[2:0]。再调用子目标生成函数,同样利用 strcmp()函数与符号表对比判断指令的寄存器号,生成各操作数的目标代码,即 inst[11:9]、inst[8:6]、inst[5:3]。在指令处理过程中进行语法分析,发现语法有错误就调用 err 函数。本文设计冒泡算法为测试用例,具体汇编代码如下所示:

```

LDIMI rd0 001001 //立即数 9 送 R0
SUBI rd0 rd1 000001 //R0 中数据(9)减 1 送 R1
LDM rd0 rd3 000000 //以 R0 中数据为地址,在
RAM 中的数据送 R3
LDM rd1 rd4 000000 //以 R1 中数据为地址,在 RAM
中的数据送 R4
ADD rd6 rd6 rd6 //延时槽
ADD rd6 rd6 rd6 //延时槽
SUB rd4 rd7 rd3 //R4 中数据减 R3 中数据送 R7
BGZ rd7 000000111 //R7 中数据大于 0,跳转到 PC+7
地址
BEZ rd7 000000110 //R7 中数据等于 0,跳转到 PC+6
地址
MOV rd3 rd5 //R3 数据送 R5
MOV rd4 rd3 //R4 数据送 R3
MOV rd5 rd4 //R5 数据送 R4
  
```

```

STM rd0 rd3 000000 // R3 中数据送到以 R0 中数据
为地址的 RAM 中
STM rd1 rd4 000000 // R4 中数据送到以 R1 中数据
为地址的 RAM 中
SUBI rd1 rd1 000001 //R1 自减去 01H
BGZ rd1 111110011 //R1 中数据大于 0,跳转到 PC—
13 地址
BEZ rd1 111110010 //R1 中数据等于 0,跳转到 PC—
14 地址
SUBI rd0 rd0 000001 //R1 自减去 01H
BGZ rd0 111101111 //R0 中数据大于 0,跳转到 PC—
16 地址
ADD rd6 rd6 rd6 //延时槽
HLT //停机
  
```

5 仿真综合结果及分析

5.1 Modelsim 仿真

在 Modelsim 中编写 testbench,进行仿真调试,ROM 中读入数据,完成对数据从高地址开始由小到大进行排序。如图 3 仿真时序图所示,本设计解决了数据和控制相关性问题,在处理发生分支跳转指令时,预测下一条指令就是要执行的指令,顺序取指。在 ID 级进行条件判断并且计算下一条指令的地址,控制器判断条件是否满足。信号 stopid 为 0 时,发生跳转,则硬件预测的指令错误,控制器在 ID 级将此指令废弃,用空指令代替 IF 级传输过来的指令,IF 级根据计算的跳转地址取指;当 stopid 为 1 时,判断不发生跳转,指令顺序执行。

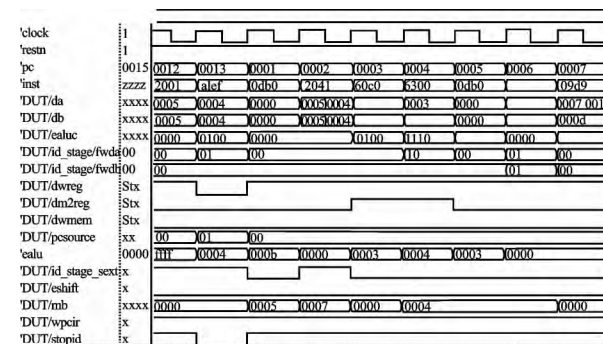


图 3 时序仿真

流水线的各阶段在 clock 的上升沿开始执行。为避免在时钟上升沿对寄存器同时读写造成数据冲突,在上升沿对寄存器进行读操作,在下降沿写回寄存器堆。实验验证表明 RAM 中所有数值排序正确完成,最后执行一个 HALT 停机指令。

5.2 实验结果对比

如图 4、图 5 所示,描述了 CPU 运行测试程序完毕时的仿真结果。图 4 是该 CPU 使用了硬件预测技术的仿真图,图 5 是传统使用延时槽仿真所产生

的时序图,如图中所示,相同的测试程序,相同的时钟周期,本文设计的 CPU 大约使用 7 620 ns 时间执行完测试冒泡算法,而如图 5 所示,通过增加延时槽的方法解决控制相关问题,明显增多了 CPU 的执行周期,实验需要 8 628 ns 的时间执行完毕. 本文所设计 CPU 明显减少 CPU 执行所需时钟周期数,加速了 CPU 的处理速度,相对缩短了 11.68% 的执行周期.

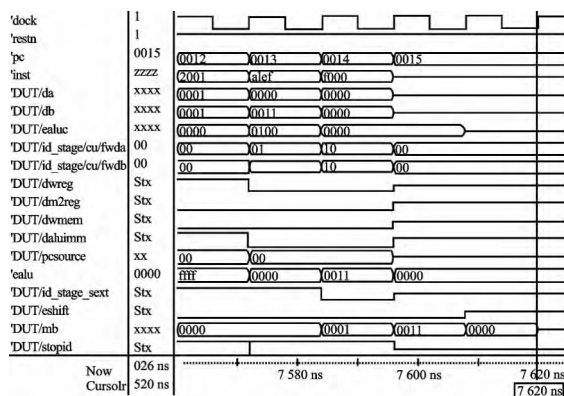


图 4 硬件预测法

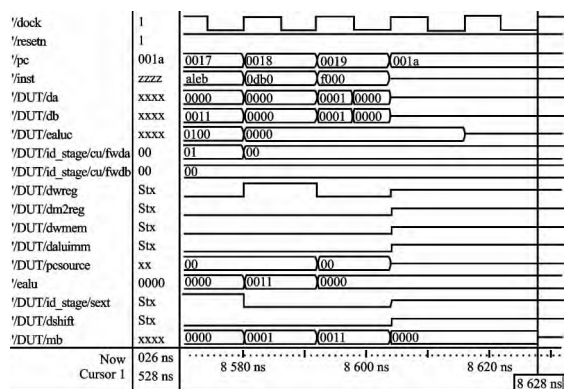


图 5 跳转延时法

本文使用 Quartus 进行软件综合,用 CycloneII 型号 FPGA 库进行仿真综合,首先将各模块分别进行综合,最后对整个模块进行综合. 综合结果显示最

小的周期是 16.078 ns,所能达到的最大时钟频率是 57.49 MHz,关键路径的时间是 17.394 ns. 整个处理器资源使用占 FPGA 总资源不到 14%.

6 结束语

本文设计的 16 位流水线 CPU^[7],主要解决流水线带来的数据相关和控制相关问题,使用旁路技术建立到 ID 级的专用通路,把计算的结果或读出的数据提前推送到 ID 级来解决数据相关;关于跳转指令,一方面硬件预测其后续一条指令,顺序取指,另一方面提前判断跳转条件以及计算跳转的地址,从而解决控制相关. 在软件架构上,设计指令集相应的汇编器. 整个 CPU 的架构用 Verilog 语言实现,并在 QuartusII、Modelsim 完成综合仿真,验证了该 CPU 的设计符合要求.

参考文献:

- [1] 郑纬民,汤志忠. 计算机系统结构[M]. 2 版. 北京:清华大学出版社,2003.
- [2] 李亚民. 计算机组成原理与系统结构[M]. 北京:清华大学出版社,1999.
- [3] 黄望华,李坤雷,刘怡俊,等. 基于 FPGA 流水线 RISC 微处理器的设计[J]. 微计算机信息,2008,24(5-2): 187-189.
- [4] 赵刚,张垒,高翔戈,等. 计算机系统的自主设计[M]. 北京:电子工业出版社,2011.
- [5] David A Patterson, John L Hennessy. Computer organization and design[M]. [s. l.]: China Machine Press,2006.

作者简介:

袁 婷 女,(1990-),硕士研究生. 研究方向为计算机体系结构,嵌入式系统. E-mail:982346404@qq.com.

刘怡俊 男,(1977-),博士,教授. 研究方向为嵌入式系统及片上网络.