

基于哈佛结构的单周期堆栈处理器设计

何凯成, 施慧彬

(南京航空航天大学 计算机科学与技术学院, 江苏 南京 210016)

摘 要: 本文设计了一种基于 Forth 语言的堆栈式 16 位单周期处理器, 该处理器含有深度为 32 的数据栈、返回栈和哈佛存储结构, 其指令集包括 29 条指令, 它的所有指令均在单个周期内执行完成。此处理器在 Xilinx 公司的 XC5VLX110T FPGA 开发板上实现的最高时钟频率为 162.7MHz。本文首先介绍了此处理器的设计方法、整体结构和指令集, 然后分析指令单周期执行的机制, 最后给出了处理器的仿真和综合结果及其分析。

关键词: Forth; 堆栈; 哈佛结构; FPGA

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2013)09-0066-05

Harvard Structure—Based Single Cycle Stack Processor Design

HE Kai-cheng, SHI Hui-bin

(Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics,
Nanjing 210016, China)

Abstract: A 16-bit single cycle stack processor based on Forth programming language is proposed in this paper, the processor has a data stack and a return stack of 32 elements deep, and a Harvard structure storage system. The instruction set of it includes 29 instructions, all of which could be completed within single cycle. When implemented on Xilinx's XC5VLX110T FPGA, The processor could run up to 162.7MHz. The paper first introduces the design principle, architecture and instruction set of the processor and then analyses the strategy of instruction single cycle execution. Finally, it presents the result of hardware synthesizing and software simulation and analysis of it.

Key words: Forth; stack; Harvard structure; FPGA

1 引言

计算机界关于处理器的 RISC(Reduce Instruction Set Computer)^[1]和 CISC(Complex Instruction Set Computer)^[2]体系结构的争议已有很长的历史, 而这一争议最终以 RISC 结构的胜利而结束。

Forth 语言是一种高度模块化、可扩展的、基于堆栈的语言^[3], 其堆栈处理器从一定程度上解决了 RISC 结构处理器存在的问题。

在 Forth 语言的程序设计过程中会涉及大量的子程序调用。文献[4]提到, 子程序调用指令在 Forth 程序中的使用比例为 25%, 加快子程序调用指令的执行速度将有利于处理器性能的提升。因此本文根据 Forth 语言定义的处理器体系结构设计了

一个具有高速子程序调用的含有 29 条指令的 16 位堆栈处理器。该处理器具有分离的指令和数据存储器、32 个元素深度的数据堆栈和返回堆栈, 它的所有指令均是零操作数的单周期指令。

2 处理器结构与分析

2.1 处理器结构

图 1 为处理器的结构图。图中与实心圆点相接的多条线段是连接在一起的, 相交处没有实心圆点的两条线表示这两条信号线没有连接在一起, 用实心箭头表示的信号是处理器控制信号。其中处于最外围矩形框内部的信号为处理器内部信号, 而伸出最外围矩形框的信号为处理器与外界接口信号。图 1 中, “clk”和“rst”为处理器的时钟信号和复位信

号;“pc”和“im_data”为处理器与指令存储器的接口,“pc”是指令存储器访存地址,“im_data”是指令存储器数据线;“dm_data”、“wr_data”、“dm_adr”和“dm_wr”是处理器和数据存储器的接口,“dm_data”是数据存储器读出数据,“wr_data”是数据存储器写入的数据,“dm_adr”是数据存储器访存地址,“dm_wr”是数据存储器写使能信号. 处理器指令存储器的内容不会被处理器修改,所以它被设计为一个只读存储器,而数据存储器是可读写存储器.

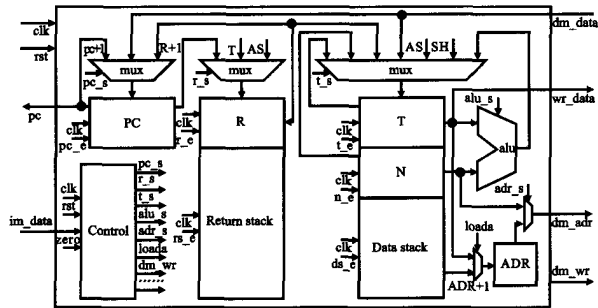


图 1 处理器结构图

如图 1 所示,处理器的主要模块有:程序计数器 PC、控制逻辑 Control、返回栈栈顶寄存器 R、返回堆栈 Return stack、数据栈栈顶寄存器 T、数据栈次顶寄存器 N、数据堆栈 Data stack、算术逻辑单元 alu、数据存储器地址寄存器 ADR. T、N 和 Data stack 共同组成数据堆栈,其中 T 和 N 分别表示数据栈的栈顶元素和栈顶下的第一个元素(也称为次栈顶元素),Data stack 为次栈顶下的元素寄存器堆. 数据栈用于存储算术逻辑运算指令的操作数和访存指令的访存地址和数据. 返回栈由 R 寄存器和 Return stack 组成,它主要用于存储子程序调用时的返回地址.

2.2 指令集

如表 1 所示,本文堆栈处理器共有 29 条指令,可分为四类指令:算术逻辑运算、堆栈调整、访存、程序转移.

上述大部分指令的功能与 Forth 语言完全一致,具体指令功能可参考文献[4]的附录部分,部分不一致的特殊指令在下文中特别提出并说明.

算术逻辑运算指令只会对数据堆栈的栈顶前两个元素进行处理,如:AND 指令执行时会先将 T、N 的内容作为 ALU 的操作数进行与运算,然后将 ALU 的运算结果送入 T 中,将 N 中原来的数据丢弃,并从 Data stack 中弹出一个元素送入 N 中. 其中,一部分算术逻辑运算指令只会对 T 中的内容进行处理,如:--,SHL,SHR.

表 1 处理器指令集

分类	指令名称
算术逻辑运算	AND、OR、XOR、+、-、--、MPP、SHLD、<、>、=、SHL、SHR
堆栈调整	R>、>R、DUP、OVER、DROP、SWAP、NOP
访存	@、!、LIT
程序转移	CALL、RET、JMP、JZ、JNZ、LOADA

本文堆栈处理器对算术逻辑运算指令进行了简化,去掉了不常用的指令,取代的是其它的指令的组合序列,这些组合序列完成的功能和相应的不常用指令的功能是完全一致的. 如:在文献[3,5]中的堆栈处理器都添加了一 ONE、ZERO 等指令,其实这些指令在常用的嵌入式程序中使用的频度并不高. 如果将它们加入处理器的指令集中,只会对该处理器的编译器的设计带来很小便利. 而对于处理器来说,这将会增加其控制器译码逻辑的复杂度,从而增加每一条指令的译码时间,对处理器性能的提高没有好处.

堆栈调整指令的操作对象是返回栈的栈顶 R、数据栈的栈顶 T 和次栈顶 N. 其中 NOP 指令为空操作指令,它不会改变处理器的状态. DUP、OVER、DROP、SWAP 指令是数据堆栈调整指令,DUP 和 OVER 指令会产生数据栈压栈操作,SWAP 将交换 T 和 N 中的数据.>R 和 R> 指令会在数据栈和返回栈间转移栈顶数据.

堆栈处理器中的堆栈调整指令和算术运算指令中,指令的默认操作数均来自于数据栈顶的前两个栈顶元素. 在文献[3]中,将数据栈的前三个元素独立了出数据栈. 在分析了文献[3]处理器的指令集后,发现只有少数不常用指令会引用到数据栈的第三栈顶元素,如:ROT、MPP 和 SHLD. 因此,本文堆栈处理器去除了 ROT 指令,并对 MPP、SHLD 进行了改造,以使处理器的第三栈顶元素不需要独立出来,节省了宝贵的硬件资源. ROT 指令的具体功能是将栈顶、次栈顶和第三栈顶元素进行循环交换,如:当栈顶前三个元素分别为 A、B、C,A 为栈顶,则执行 ROT 操作后,栈顶前三个元素为 C、A、B. ROT 指令可以由指令序列“>R SWAP R> SWAP”来完成. 在文献[3]和文献[5]中 MPP 和 SHLD 指令均要借助于栈顶、次栈顶和第三栈顶来完成,而本文处理器 MPP 和 SHLD 指令使用返回栈栈顶、数据栈顶和次栈顶来完成,这样增加了处理器的资源利用率. 其中 MPP 和 SHLD 指令是组成

乘法和除法运算的基本步骤。

为了加快处理器的访存性能,本文处理器的访存指令只设置了三条。其中@指令用于从指定地址中读出数据,!指令则相反。@和!指令的访存地址为N寄存器,@指令的目的寄存器和!指令的源寄存器为T寄存器。LIT指令是从当前数据存储器地址中取出数据压入数据栈,LIT的默认访存地址存放在ADR寄存器中。程序转移指令有三类:程序调用与返回、无条件分支和条件分支。其中,程序调用和两种分支指令的默认跳转地址均来自数据存储器。需要说明的是在程序转移指令中,指令LOADA不执行程序跳转,它的功能是将寄存器T中的内容存入地址寄存器ADR中。指令LOADA不属于Forth语言的基本操作集,它是为了实现指令单周期执行而设计的一条特殊指令,它的特殊功能将在第4节中进行说明。

2.3 指令集格式

指令集的格式对控制器的设计有很大的影响。本文对处理器的指令集编码进行了一定的优化处理,使控制信号相似度高的指令归为同一类,这样使得控制器的译码逻辑变得更加简单。

表2是指令优化分类表,指令分为A、B、C、D四类。在每个指令类的内部,指令的控制信号相似度十分高,而属于不同指令类的指令,它们的控制信号相似度比较低。在A类指令中,所有的指令的默认操作数均为T和N,所有指令的执行都要对ALU模块发出控制信号。所有A类指令都将T和N中的数据元素丢弃,将ALU执行结果压入T中,并从Data stack中弹出一个元素送入N中。如:指令AND会取出T和N中的元素,送入ALU中运算,然后将运算结果送入T中,并将N中元素丢弃。在B类指令中,所有的指令对ALU发出的控制信号都是无意义的,指令都涉及到对数据堆栈元素的移动。如:LIT会从数据存储器中取出一个元素压入数据栈,此时,T、N、Data stack的数据写使能都将会有效。在C类指令中,除去指令!和LOADA外,其它指令会对T进行写操作,而指令MPP、SHLD会同时对R和T进行写操作,而指令!和LOADA有一个共性是它们不会改变T和N中的数据。D类指令的特性十分明显,它们都是分支指令,分支指令都会改变程序计数器PC中的内容。在进行译码时,又将D类指令分为两小类:条件转移和无条件转移。其中CALL、RET、JMP为无条件转移指令,JZ和JNZ为条件转移指令。无条件转移指令会无条件的

修改PC中的内容,而条件转移以T中的内容为条件修改PC中的内容。

表2 指令优化分类

指令分类	指令名称
A类	AND、OR、XOR、+、-、>、<、=
B类	LIT、DUP、OVER、@、DROP、>R、R>、SWAP
C类	MPP、SHLD、SHR、SHL、--、!、LOADA、NOP
D类	CALL、RET、JMP、JNZ、JZ

本文处理器有29条指令,使用5个比特位表示所有指令。处理器的机器字长为16位,控制器将数据的高字节的低5位和低字节的低5位分别用来表示一条指令,每次取指都会一次性取出两条指令。上述的四种指令的分类号用两个比特位表示,然后使用剩余的三个比特来表示分类内部不同的指令。D类指令上具有一个小分类,因此将剩余三个比特的最高位拿出来作为小分类的标识。

3 控制器状态机

本堆栈处理器的一个机器字中会存储两条指令,每一条指令的执行占用一个时钟周期,所以处理器一般会每两个时钟周期取一次指令。若指令中出现程序转移指令并且程序转移指令位于指令字的高字节中时,处理器可能会在取完指令后的下一个周期继续取指令。图2为控制器状态机的状态转换图,图中的大圆表示状态机的状态,带有方向箭头的曲线表示状态间的转换,曲边上字符或标号表示状态转换条件。

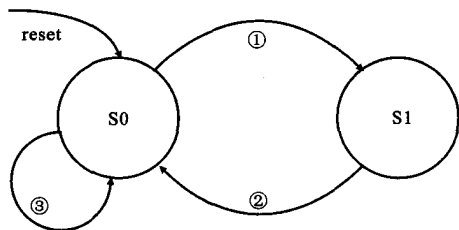


图2 控制器状态机状态转换图

状态“S0”表示处理器执行机器字高字节指令,状态“S1”表示处理器执行机器字低字节指令。处理器上电的同时,处理器复位信号有效,状态机复位为状态“S0”。在“S0”状态,如果指令字的高字节为无条件跳转指令如JMP,或者指令字的高字节为条件跳转指令且条件跳转成功时,处理器在时钟沿到来时会重新取指令,状态机会转换到“S0”状态,这对

应图2中的转换条件“③”。在“S0”状态,指令字的高字节为非跳转指令时,在执行完本周期指令后,状态机转换到状态“S1”,图2中转换条件“①”与之对应。在“S1”状态,处理器会无条件的执行取指令操作,状态机无条件的转换到“S0”状态,在图中的转换条件为“②”。

由于程序跳转地址是按机器字进行编址,所以程序转移指令跳转到的目的指令只会处于指令字的高字节。若出现在指令字高字节位置的指令为无条件跳转指令,则在此指令字的低字节中的指令将不会有执行的机会,因此这个字节中应该插入指令气泡NOP。只有在指令字高字节为非跳转指令或条件跳转指令且跳转失败的情况下,指令字的低字节才能够被执行。

4 指令单周期执行机制

在Forth基本操作集中,单操作数指令的操作数通常是与相应指令操作码存放在相邻的存储位置上。例如:指令LIT在Forth基本操作集中是单操作数指令,为了指令的取操作数方便,LIT操作码与其操作数存放在指令存储器的相邻位置。此外,这种单操作数指令的操作数存放顺序是与指令存放顺序一致的。例如:指令序列“LIT 9, CALL 0, JMP 4”中,指令的操作数9、0、4的存放顺序是与LIT、CALL、JMP的存放顺序是一致的。如果将单操作数指令的操作数从指令存储器中分离出来,按指令执行先后顺序存放在单独的数据存储器中,使这种指令在执行时具有默认操作数,则可使这些单操作数指令变成零操作数指令。在执行这些单操作数指令时,指令的操作数可以进行预取操作,这样使得指令的访存与操作数的访存操作可以同时进行。根据这种想法,本文处理器设置了一个特殊的访存地址寄存器ADR,它将Forth语言中的单操作数指令变成具有默认操作数寻址的零操作数指令。

ADR寄存器是一种类似PC的寄存器。PC寄存器每当取出一条指令,PC的内容会自动加一,只有发生指令跳转时,PC内容会被修改。ADR寄存器在每一次取出一个数据后,ADR中的访存地址会自动加一,同时它的内容也可以通过指令LOADA进行修改。在指令执行过程中,若有数据存储器访问指令在执行,则以ADR中内容为地址的数据存储器中的数据将会被送入处理器执行,同时ADR寄存器中的内容会自动加一。如:当处理器在执行LIT指令时,以ADR为地址的数据会被压入数据栈,同

时, $ADR = ADR + 1$ 。当程序中有循环结构且循环中具有数据存储器访问指令时,采用连续空间数据访问将会产生存储空间浪费问题。例如,一个循环中要执行10次LIT指令,按照ADR自动加一的操作方式,就要在数据存储器的10个连续的空间上存放10个相同的数据,这样是对存储空间的浪费。因此本文的堆栈处理器中增加了一条非Forth基本操作集的指令——LOADA,这条指令用于将T中的数据内容存入ADR寄存器中,同时T、N和Data stack进行弹栈操作。对于循环指令,可以使用LOADA指令来重复使用同一地址范围内的数据。

在其它堆栈处理器中,LIT、CALL等指令的执行一般要浪费两个时钟周期:前一个周期去访问存储器,后一个周期进行指令执行,或者前一个周期指定访存地址,后一个周期访问存储器。子程序调用指令在堆栈处理器程序中使用频率很高,加快此指令的执行性能可以提高处理器的运行性能。本文处理器中的指令LIT、CALL、JMP、JZ、JNZ指令均是默认操作数寻址指令,它们的默认访存地址都是来自ADR寄存器,它们的执行均只需要一个周期。

5 仿真与性能分析

本文处理器采用Mentor Graphics公司的仿真工具Modelsim6.5c进行仿真,图3为处理器的累加程序仿真图。其中“rst”为系统复位,“clk”为系统时钟信号,“pc”为程序计数器,“instr”为指令字,“instr_byte”为当前指令,“t”为T寄存器信号,“n”为N寄存器信号,“ds1”为Data stack的第一个元素,“ds2”为Data stack第二个元素,“r”为R寄存器信号。

此累加程序的步骤如下:首先加载累加计数和初始累加和,然后进入累加循环中,完成累加后跳出累加循环。在图3中,pc=0000和pc=0001中的四条指令用于加载累加计数和初始累加和。在25ns时刻执行完一条LIT指令后,T寄存器中的数据变为初始累加和0;25ns~45ns之间执行的指令分别为DUP和XOR指令,这两条指令功能是确保初始累加和为0;在55ns时刻执行完一条LIT指令后,累加计数4被送入T寄存器中、初始累加和被送入N寄存器中。图3中,pc=0002至pc=0006之间的指令为累加循环,其中pc=0002处的两条指令用于初始化循环指令的数据存储器访存地址。105ns时刻执行完一条>R指令,它将T寄存器中的数据转存入R寄存器中,可以看到105ns后,R寄存器内容

变为 4.115ns 时刻,处理器执行完一条加法指令,它将 T 和 N 中的内容相加形成新的部分累加和。在 125ns 和 135ns 时刻分别执行了指令 R> 和 --,它们用于将累加计数减一。在 145ns 和 155ns 时刻分别执行了指令 JZ 和 JMP,它们用于判断累加计数是否到达零。在 155ns 时刻,新一轮的循环运算开始。

本文处理器采用 Synopsys 公司的综合工具 Synplify pro 9.6.1 进行综合,它在 Xilinx 公司的

XC5VLX110T FPGA 上综合的最高时钟频率为 162.7MHz,其它同类设计的最高时钟频率为 146.7MHz^[3];它在 Xilinx 公司的 Spartan-II E FPGA 上综合的最高时钟频率为 65.7MHz,其它同类设计的最高频率为 55.9MHz^[3]。文献[3,5,6]中的处理器均为多周期处理器,而本堆栈处理器为单周期处理器。由此可见本处理器在堆栈指令处理性能上相对其它堆栈处理器有一定的提升。

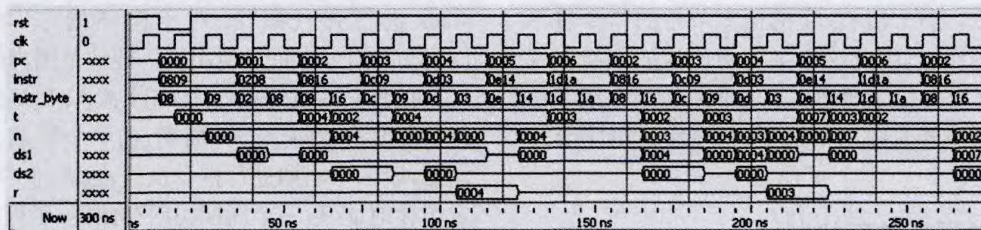


图3 累加程序仿真图

6 结束语

本文介绍了一种基于 Forth 基本操作集的具有哈佛存储结构的堆栈处理器。该处理器具有优化的编码指令集、简易的指令控制逻辑和完全的单周期指令执行能力。处理器最终在 Xilinx 公司的 XC5VLX110T FPGA 开发板上实现,最高时钟频率达到 162.7MHz,它在指令处理性能方面优于其它同类处理器。本堆栈处理器的下一步工作将是对处理器的逻辑结构进行进一步的优化,使其指令运行有更高的性能。

参考文献:

- [1] 张金钟,胡平. 基于 RISC 结构的 Java 处理器研究与设计[J]. 微电子学与计算机,2011,28(7): 61-62.
- [2] 靳文兵,左琦. 复杂指令集流水线系统设计[J]. 微电子学与计算机,2012,29(4): 19-20.
- [3] 储昭贤,施慧彬. 基于 FPGA 的 16 位堆栈处理器的设计[J]. 微电子学与计算机,2012,29(2): 24-26.
- [4] Phil Koopman. Stack Computers: the new wave [M]. New York; Chichester: E. Horwood, 1989: 59-60.
- [5] Richard E Haskell. A VHDL-Forth Core for FPGAs

[J]. Microprocessors and Microsystems, 2004, 28 (3): 115-125.

- [6] Bernd Paysan. B16 - a forth processor in an FPGA [EB/OL]. (2003-10-19) [2012-10-12]. <http://www.jwdt.com/~paysan/b16.html>.

作者简介:



何凯成 男,(1987-),硕士研究生. 研究方向为计算机系统结构。



施慧彬 男,(1966-),博士,副教授. 研究方向为计算机系统结构、可重构计算、编译优化技术。