超标量处理器设计SPEC

目录

[1. 项目背景 3](#_Toc185499017)

[2. FPGA选择 3](#_Toc185499018)

[3. 整体架构 3](#_Toc185499019)

[4. 设计细节 4](#_Toc185499020)

[4.1 TLB与cache 4](#_Toc185499021)

[4.2分支预测 4](#_Toc185499022)

[4.2.1局部历史预测 4](#_Toc185499023)

[4.2.2 BTB与RAS结构设计 5](#_Toc185499024)

# 项目背景

为了回顾巩固《超标量处理器设计》这本书，准备学习编写一个乱序执行的超标量处理器，同时这也是为了提升自己的水平。

长远目标是要在FPGA上跑通该项目，并成功运行Linux微系统，这有一定的难度。中期目标则是要完成这整个项目的代码编写，并尝试做一些验证。

如果能借此契机参加一下龙芯杯就更好了。下列链接是关于龙芯杯的一些信息：

[【CQUT】从零开始龙芯杯-初章 - 哔哩哔哩](https://www.bilibili.com/opus/993344116458782775)

# FPGA选择

初步准备选择Xilinx AC7010系列-XC7Z010，关键参数有：512MB的DDR3和256KB的BRAM。更详细的细节参见：

[ALINX FPGA SOM AC7010: with AMD Zynq7000 SoC FPGA module Industrial Grade](https://china.xilinx.com/products/boards-and-kits/1-t5lo2d.html)

# 整体架构

本次设计主要参考《超标量处理器设计》中page346的流水线架构图，一些细节的内容有待进一步商榷、修改。



系统整体采用32位，支持MIPS32的一些基础指令，可以参加D:\个人\代码积累\Verilog\超标量处理器设计 目录下《MIPS\_Vol2》。

# 设计细节

## 4.1 TLB与cache

支持TLB，采用32位的虚拟地址，32位的物理地址。因此理论上来说DRAM可以支持到1GB。采用4KB的小页形式，也就是地址的[11:0]是虚拟地址与物理地址共有的。为了便于内存空间占用，TLB缺失时虚拟地址到物理地址的转换设计为三级结构：第一级用ASID，第二级用虚拟地址[31:22]，第三级用虚拟地址[21:12]，访问内存三次方可获得相应物理地址。

TLB设计分为I-TLB和D-TLB，每个TLB都由32级全相联的结构构成，TLB的缺失设计为硬件自动完成访问内存，如果在内存里再次发生缺失将引发page-fault

cache支持虚拟存储器，并选用virtual-index+pysical-tag的形式，L1-cache两级流水线访存使用虚拟地址，第一级流水取出cache内容以及TLB的物理地址，第二级流水完成相应的tag比对。

cache的容量设计上，L1-cache分为I-cache和D-cache，为了方便virtual-index，用地址的[11:0]来进行索引，这样只能实现4KB的容量，在此基础上实现4路组相联，这样可以使得L1-cache总容量达到16KB+16KB共计32KB。L2-cache使用纯物理地址，预设容量为128KB，暂时不准备采用组相联的结构。关于cacheline的大小，选择64byte即512bit。

## 4.2分支预测

预计采用历史预测的策略。

### 4.2.1局部历史预测

局部历史预测：参考《超标量处理器设计》page111图4.17，不过放弃hash处理。对于PC[31:0]，[1:0]永远=0。利用PC[7:2]去寻址BHT，可见BHT的表项有64个，且每个表项记录的BHR为6bit，即记录某一条分支指令过去六次的分支情况。最后用BHR+ PC[5:2]这样一个10bit的数去寻址PHT，得到PHT中两位饱和计数器的值，完成预测。

在四decode-width的超标量处理器中，每次要实现对四条指令的PHT查询，所以对BHR以及PHT都要使用4BANK的堆叠设计，由于访问PHT的地址是BHR+ PC[5:2]，访问BHR的地址是PC[7:2]，四条相邻指令的最后2bit必然是不一样的，可以完美地实现交叠访问。

局部历史预测所需的BHT：2^6\*6bit（其中2^6分为2^4\*4），局部历史预测所需的PHT：2^10\*2bit（其中2^10分为2^8\*4）

### 4.2.2 BTB与RAS结构设计

BTB的结构与cache类型，参考P124图4.32，但是有一处细节需要明确，基于SRAM的cache其时序是：第一拍给出访问信号，第二拍给出结果。这在CPU的流水线中无法满足要求，PC寄存器要在每一拍都完成相应的变化。因此BTB结构的底层器件应该基于寄存器堆/特制的SRAM。

为了应对超标量处理器的设计，预设的取指模式是在4字对齐的边界，所以我们使用PC[31:4]去访问BTB。这里有两个重要的前提，第一是BTB里只记录了对应4条指令中一条分支指令的跳转地址，所以我们预设4字中最多只有一条分支指令；第二是由于取值总是在4字对齐的边界，有时候我们可能无法一次取出四条指令，每次取出0123，如果给的地址指向0那我们就将0123都取出，但有可能给的地址指向2，那我们即时0123都取出也只能使用23。这很重要。

BTB的结构预设在128个表项，使用PC[10:4]，tag项为PC[31:11]，除此之外BTB的内容还包含一个vld项、2bit表明分支指令类型、4bit表明分支指令在该组指令中的位置以及32bit的目标地址项。预计容量所需：2^7\*(1+2+21+4+32) bit≈8Kb≈1KB

RAS选择栈结构，先进后出，他的时序需要和BTB同样，也就是能在一拍内给出结果，预设容量在2^6\*32bit=2Kb，也就是最多可以存放64个最近的return指令的目标地址。

## 4.3指令解码

解码宽度设计在4条指令，也就是4-way超标量处理器。上一个取指令环节，我们约定取值地址默认是四字对齐的，如果因为跳转指令或者其他指令使得PC的数值不再四字对齐，而是落在了四字中，那么取出的指令只有PC及其后方的指令有效。

在解码单元前没有设计Instruction-buff，取出四条指令直接送往解码单元，对于乘法以及乘加这样的有两个目标寄存器的特殊指令，采取的策略是《超标量处理器设计》page184中图6.4的方法，暂停解码之前的流水线，等待两次解码完成。

指令解码要做的事有两件：分析出源寄存器以及目标寄存器提供给重命名单元；保留指令中其他的重要信息，如执行什么操作，立即数是多少等等。