

计算式组成原理

单周期 CPU 设计

王天一,tianyiwang58@gmail.com

王天志,wangtzh20@lzu.edu.cn

聂嘉一,niejy20@lzu.edu.cn

叶清扬,yeqy20@lzu.edu.cn

2022 年 4 月

摘 要

在这次作业中，我们设计了一个基于 MIPS 指令集的单周期 CPU。
并对于我们的成品，进行了进一步验证：

- 1) 进行了功能前仿。
- 2) 测试了性能指标（频率、已实现指令的条数和 CPI）。
- 3) 交叉编译简单的 C 程序，使用 MARS 运行测试。
- 4) 使用了 FPGA 静态存储器（俗称 IP 核）存储机器指令。

关键词：CPU, Verilog, 计算机组成原理

目录

1	引言	1
2	指令集	1
2.1	段落	1
2.2	字体	2
2.2.1	文字强调	2
2.2.2	子小节	2
2.3	公式	2
2.4	定理	2
3	实验	2
3.1	图	3
3.2	表	3
4	性能指标分析	4
4.1	CPI	4
4.2	等效频率	4
5	代码实现	4
6	数据通路	4
7	加分项一	4
7.1	技术实现	4
7.2	运行截图	5
8	加分项二	5
9	总结	5
	参考文献	6

1 引言

这个模板是 UTF-8 编码的，使用 xeCJK 宏包，中英文混排更美观，但编译速度稍慢。注意：该模板只能用 xelatex 编译。本模板支持 Overleaf 在线 \LaTeX 平台，网址：<https://www.overleaf.com>。详细的 \LaTeX 使用方法这里不详细展开，仅列举一些常用的。

引言也称前言、序言或概述，作为科技论文的开端，提出文中要研究的问题，引导读者阅读和理解全文。

2 指令集

对于指令集，我们参考了 MIPS 指令集共实现了 **31** 条指令集，其中包括 **17** 条 R 型指令，**12** 条 i 型指令，**2** 条 J 型指令。

2.1 段落

[illegible]

这是下一段。假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落，假设这是一个很长的段落。

这是另一段

这又是一段。

这是最后一段，它由好多好多非常多好多非常多好多非常多好多非常多好多非常多好多非常多好多非常多好多非常多的条目组成，就像下面这个样子：

- 这里引用了一篇文献 [1]。
- 这是一个很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很

长很长的条目。

2.2 字体

默认字体为宋体。**这是黑体。**这是宋体。 这是仿宋。 这是楷体。或者**黑体**，宋体，仿宋，楷体。

2.2.1 文字强调

加粗宋体：**粗体**，加斜字体自动变成楷体：*强调*。

更多中文说明 (网址有点长, 显示不全):

<https://www.overleaf.com/latex/examples/using-the-ctex-package-on-overleaf-zai-overgndvpvsmjcqx/viewer.pdf>

2.2.2 子小节

这是另一个子小节。

2.3 公式

$$\mathbf{A} = \sum_{i=0}^{N-1} e^{j\frac{2\pi}{N}i} = 0 \quad (1)$$

2.4 定理

正文中可能有很多定理、定义、证明、引理等等。这一特性目前可以手动实现和标号，以后会添加自动方法。

3 实验

[illegible]

[illegible]

3.1 图

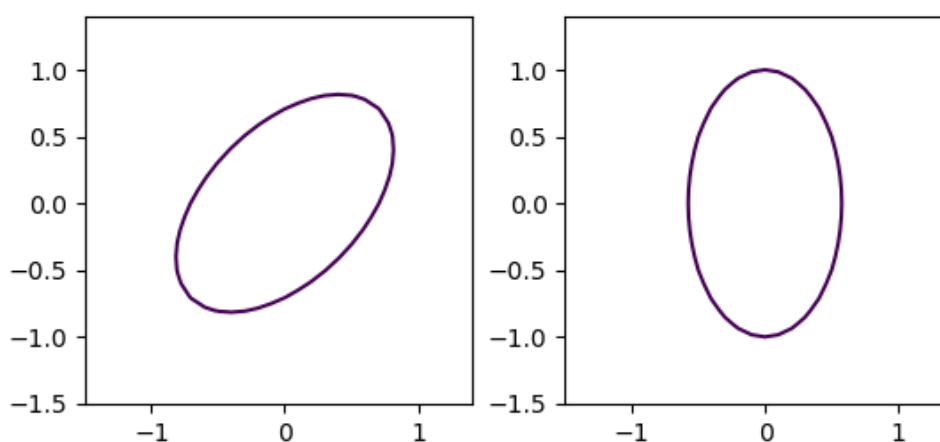


图 1: 这是一个图

引用图 1。

3.2 表

表 1: 这是一个表

	卡尔曼滤波	神经网络滤波	被动无源滤波
模型类型	线性	线性	非线性
参数调校	大量	几乎没有	合理
稳定性	满足全局稳定性	依赖于模型	满足子系统稳定性

引用表格 1。

4 性能指标分析

4.1 CPI

参考课件，我们可以知道，CPI 计算公式为

$$CPI = \frac{\text{CPU 时钟周期数}}{\text{指令条数}}$$

显而易见，在我们的单周期 CPU 上，一个时钟周期只运行一条指令，因此我们可以计算出 $CPI = 1$ 。

4.2 等效频率

参考课件，我们可以知道等效频率的计算公式为

$$\text{等效频率} = \frac{1}{\text{时钟频率}}$$

5 代码实现

6 数据通路

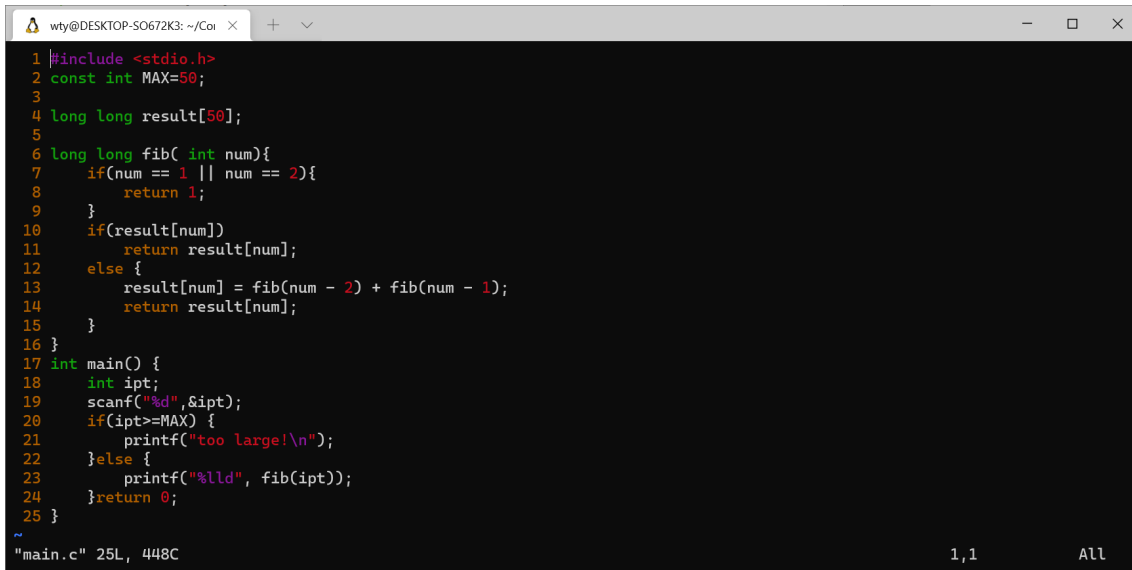
7 加分项一

使用 C 语言编写简单程序，覆盖到 CPU 所支持的分指令，使用 MIPS 或 RISC-V 交叉编译 (mips-linux-gcc 或 gcc-riscv) 为汇编源码，使用 MIPS 或 RISC-V 模拟器翻译为机器指令并执行，加 0-15 分。

7.1 技术实现

这里我们测试的代码为优化过的递归版斐波那契数列，代码详见图 2。然后我们使用 mips-linux-gcc 工具交叉编译，见图。

编译完之后，我们需要对一些内容进行修改，比如 1) printf 函数和 scanf 函数转换成汇编后存在字符串解析，拼接等等操作，导致过于复杂，我们需要手动将写代码写为调用 POSIX 规范的输入输出接口。2) 编译出来的汇编代码是分别用堆栈寄存器 sp, gp 指向堆栈来存储全局变量（一般在 data 段（可以不严谨的理



```
1 #include <stdio.h>
2 const int MAX=50;
3
4 long long result[50];
5
6 long long fib( int num){
7     if(num == 1 || num == 2){
8         return 1;
9     }
10    if(result[num])
11        return result[num];
12    else {
13        result[num] = fib(num - 2) + fib(num - 1);
14        return result[num];
15    }
16 }
17 int main() {
18     int ipt;
19     scanf("%d",&ipt);
20     if(ipt>=MAX) {
21         printf("too large!\n");
22     }else {
23         printf("%lld", fib(ipt));
24     }return 0;
25 }
```

"main.c" 25L, 448C 1,1 All

图 2: 加分项一 C 语言源代码

解为堆段)) 和局部变量 (一般在栈段), 我们在 Mars 上运行的时候, 需要手动指定这些变量。

7.2 运行截图

在处理好各种问题之后, 我们获得了一个名为 “fibWithCache.asm” 的文件, 如图 3 所示。

然后在 Mars 直接打开这个文件, 运行。

8 加分项二

9 总结

这里总结全文。

参考文献

- [1] Rongmei Cao. Matrix Theory. Nanjing University of Aeronautics and Astronautics, 2017.
- [2] Perozzi, Bryan, R. Al-Rfou, and S. Skiena. "DeepWalk: online learning of social representations." (2014):701-710.
- [3] 李彦冬, 郝宗波, and 雷航. "卷积神经网络研究综述." 计算机应用 36.9(2016):2508-2515.
- [4] 雷思磊, 自己动手写 CPU


```
1|.data
2 start: .ascii "\n Input N = "
3 last: .ascii "\n The result is "
4 out1: .ascii "\n N is not legal\n"
5 out2: .ascii "\n overflow\n"
6 empty: .ascii " "
7 Hex: .ascii " 0XXXXXXXX\n"
8 buf: .word 1,1
9 .space 4096 #设定数组大小
10 .text
11 main: la $a0,start
12 li $v0,4
13 syscall #输出start
14 la $a0,buf
15 move $a1,$a0 #a1是地址 ==s2
16 li $v0,5
17 syscall #输入N
18 addi $v0,$v0,-1
19 move $a0,$v0 #a0是计数器N ==s3
20 bltz $a0,out_1
21 move $t5,$a1
22 move $t6,$a0
23 li $s1,1
24 jal FIB #按要求存储数据
25 addi $a3,$a0,0
26 li $a2,0
27 la $a0,start
28 li $v0,4
29 syscall
30 jal print #按要求打印数据
31 li $v0,10 #退出
32 syscall
33 FIB: ble $t6,$s1,ret #t6<=1就返回
34 addi $a2,$t5,4
35 lw $t1,($t5)
36 lw $t2,($a2)
37 addu $a3,$t1,$t2
38 addi $t4,$0,-1 #f3=f1+f2
39 subu $t3,$t4,$t1
40 bltu $t3,$t2,out_2 #判断溢出
41 sw $a3,4($a2)
42 addi $t5,$t5,4
43 addi $t6,$t6,-1
44 b FIB
45 print: bgt $a2,$a3,ret
46 move $a0,$a2
47 li $v0,1
48 syscall #打印下标
49 la $a0,empty
50 li $v0,4
51 syscall #空格
52 lw $a0,($a1)
53 li $v0,1
54 syscall #打印十进制数值
55 la $a0,empty
56 li $v0,4
57 syscall #空格
58 lw $a0,($a1)
59 addi $sp,$sp,-16
60 sw $ra,12($sp)
61 sw $a0,0($sp)
62 sw $a2,4($sp)
63 sw $a1,($sp) #用堆栈存储来保护寄存器中的值
64 jal hex #调用hex函数转换成十六进制
65 lw $ra,12($sp)
66 lw $a0,0($sp)
67 lw $a2,4($sp)
68 lw $a1,($sp) #恢复相应寄存器的值
"fibWithCache.asm" 103L, 3025C
```

图 3: 处理过的汇编代码截图