Y86 Pipeline Simulator

章凌豪

June 25, 2015

Outline I

开发语言	JavaScript / HTML / CSS
浏览器环境	Chrome / Firefox / Safari
第三方库	jQuery
	Bootstrap
	FileSaver.js
	Chart.js

- 不适合前后端分离

开发语言	JavaScript / HTML / CSS
浏览器环境	Chrome / Firefox / Safari
第三方库	jQuery
	Bootstrap
	FileSaver.js
	Chart.js

- 不适合前后端分离
- 配合 HTML 和 CSS，数据表现能力出色

开发语言	JavaScript / HTML / CSS
浏览器环境	Chrome / Firefox / Safari
第三方库	jQuery
	Bootstrap
	FileSaver.js
	Chart.js

- 不适合前后端分离
- 配合 HTML 和 CSS，数据表现能力出色
- 人生苦短，我用脚本语言

基本功能

- 实现了 Y86 指令集中的所有指令。

基本功能

- 实现了 Y86 指令集中的所有指令。
- 实现了流水线控制逻辑。

基本功能

- 实现了 Y86 指令集中的所有指令。
- 实现了流水线控制逻辑。
- 支持载入和解析.yo 文件，并能将每个周期内流水线寄存器的数值作为输出保存到文件。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。
- 实现了 **Y86** 汇编器和反汇编器，从而能够接受.ys 文件作为输入，也能保存汇编和反汇编的结果。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。
- 实现了 **Y86** 汇编器和反汇编器，从而能够接受.ys 文件作为输入，也能保存汇编和反汇编的结果。
- 内存的实现采用了分页技术。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。
- 实现了 **Y86** 汇编器和反汇编器，从而能够接受.ys 文件作为输入，也能保存汇编和反汇编的结果。
- 内存的实现采用了分页技术。
- 模拟实现了一个 **L1** 缓存的简化版。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。
- 实现了 **Y86** 汇编器和反汇编器，从而能够接受.ys 文件作为输入，也能保存汇编和反汇编的结果。
- 内存的实现采用了分页技术。
- 模拟实现了一个 **L1** 缓存的简化版。
- 程序运行结束后可以生成性能分析，包括对缓存命中和 CPI 的统计。

扩展功能

- 支持步进、步退、自动运行、暂停等操作，并能以不同频率（最高 1000Hz）运行。
- 显示内存中的数据并指示当前栈的位置，也可以监视一个指定的内存地址。
- 实现了 **Y86** 汇编器和反汇编器，从而能够接受.ys 文件作为输入，也能保存汇编和反汇编的结果。
- 内存的实现采用了分页技术。
- 模拟实现了一个 **L1** 缓存的简化版。
- 程序运行结束后可以生成性能分析，包括对缓存命中和 CPI 的统计。
- 添加了一条新指令 iaddl，并归纳了添加指令的步骤。

内核

- 封装 ALU、CPU，提供清晰的接口

内核

- 封装 ALU、CPU，提供清晰的接口
- 数值传递、检查条件、读写内存

内核

- 封装 ALU、CPU，提供清晰的接口
- 数值传递、检查条件、读写内存
- 对照 HCL 实现流水线控制逻辑

内核

- 封装 ALU、CPU，提供清晰的接口
- 数值传递、检查条件、读写内存
- 对照 HCL 实现流水线控制逻辑
- 编写测试用例来检查正确性

内存

- Int8Array

内存

- Int8Array
- 使用分页技术

内存

- Int8Array
- 使用分页技术
- 内存初始化为 0xCC

内存

- Int8Array
- 使用分页技术
- 内存初始化为 0xCC
- 多类型接口

缓存

- 模拟实现了一个 **L1 Cache** 的简化版，$S = 2, E = 2, B = 64, m = 32$。

缓存

- 模拟实现了一个 **L1 Cache** 的简化版，$S = 2, E = 2, B = 64, m = 32$。
- 选取策略：由于数据的读写以 32 位为单位，所以两个相邻地址之间差为 4，因此取 *index* 为地址的低第 4 位，*offset* 为地址的低第 3 位，这样可以较好地利用 **Spatial Locality**。

缓存

- 模拟实现了一个 **L1 Cache** 的简化版，$S = 2, E = 2, B = 64, m = 32$。
- 选取策略：由于数据的读写以 32 位为单位，所以两个相邻地址之间差为 4，因此取 *index* 为地址的低第 4 位，*offset* 为地址的低第 3 位，这样可以较好地利用 **Spatial Locality**。
- 抛弃策略：当两条 Line 的 *Valid* 位都为 1 时而又需要写入新数据时，选择访问次数少的一条 Line 写回内存。这样可以较好地利用 **Temporal Locality**。

缓存

- 模拟实现了一个 **L1 Cache** 的简化版，$S = 2, E = 2, B = 64, m = 32$。
- 选取策略： 由于数据的读写以 32 位为单位，所以两个相邻地址之间差为 4，因此取 *index* 为地址的低第 4 位，*offset* 为地址的低第 3 位，这样可以较好地利用 **Spatial Locality**。
- 抛弃策略： 当两条 Line 的 *Valid* 位都为 1 时而又需要写入新数据时，选择访问次数少的一条 Line 写回内存。这样可以较好地利用 **Temporal Locality**。
- 写回策略： 当一条 Line 的 *Dirty* 位为 1，即数据被修改过时，才将其写回内存。

汇编 & 反汇编

- 正则、转换、拼接

汇编 & 反汇编

- 正则、转换、拼接
- 良好的封装、低耦合

汇编 & 反汇编

- 正则、转换、拼接
- 良好的封装、低耦合
- 方便添加新指令

测试用例

#	描述
1	测试样例，包括.long、循环和调用
2	测试模拟器是否能能正确结束运行
3	测试模拟器是否能处理 Forward 优先级
4	测试模拟器是否能处理 Load/Use Hazard
5	测试模拟器是否能处理 Combination A
6	对链表中的三个元素进行求和，测试内存取值相关操作
7	同上，但用递归完成，测试调用和返回等操作