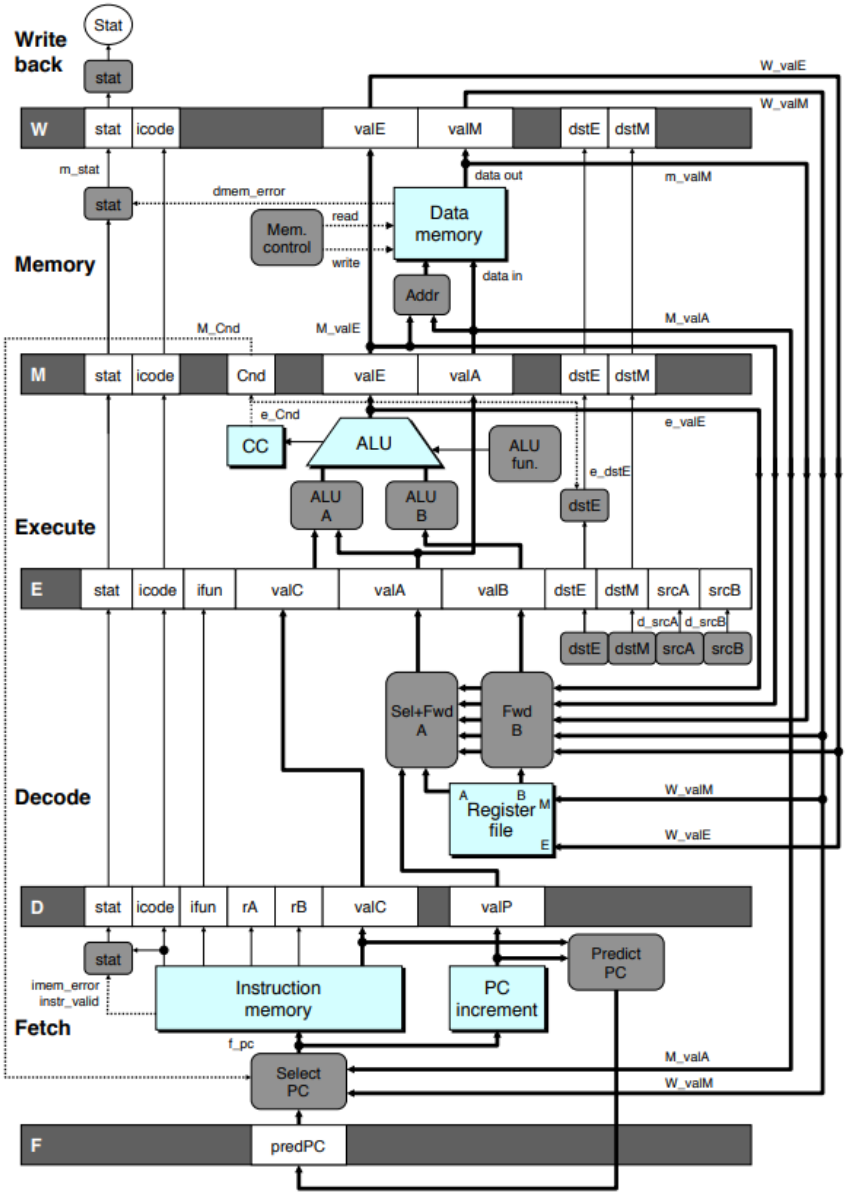
CS:APP3e Guide to Y86-64 Processor Simulators

CS：APP3e Y86-64 处理器模拟器指南



This document describes the processor simulators that accompany the presentation of the Y86-64 processor architectures in Chapter 4 of Computer Systems: A Programmer's Perspective, Third Edition. These simulators model three different processor designs: SEQ, SEQ+, and PIPE.

本文档描述了第三版《计算机系统：程序员的视角》第 4 章中 Y86-64 处理器体系结构演示的处理器模拟器。这些模拟器为三种不同的处理器设计建模：SEQ、SEQ+和 PIPE。

1 Installing

1 安装

The code for the simulator is distributed as a tar format file named sim.tar. You can get a copy of this file from the CS:APP3e Web site (csapp.cs.cmu.edu).

模拟器的代码以名为 sim.tar 的 tar 格式文件分发。您可以从 CS:APP3e 网站（csapp.CS.cmu.edu）获取此文件的副本。

With the tar file in the directory you want to install the code, you should be able to do the following:
使用要安装代码的目录中的 tar 文件，您应该能够执行以下操作：

```
linux> tar xf sim.tar
linux> cd sim
linux> make clean
linux> make
```

By default, this generates GUI (graphic user interface) versions of the simulators, which require that you have Tcl/Tk installed on your system. If not, then you have the option to install TTY-only versions that emit their output as ASCII text on stdout. See file README for a description of how to generate the GUI and TTY versions.
默认情况下，这会生成模拟器的 GUI（图形用户界面）版本，这需要在系统上安装 Tcl/Tk。如果没有安装，那么您可以选择安装仅 TTY 版本，这个版本在标准输出上以 ASCII 文本显示。有关如何生成 GUI 和 TTY 版本的描述，请参阅文件 README。
The directory sim contains the following subdirectories:
目录 sim 包含以下子目录：
misc Source code files for utilities such as YAS (the Y86-64 assembler), YIS (the Y86-64 instruction set simulator), and HCL2C (HCL to C translator). It also contains the isa.c source file that is used by all of the processor simulators.
misc 实用程序的源代码文件，例如 YAS（Y86-64 汇编程序）、YIS（Y86-64 指令集模拟器）和 HCL2C（HCL-to-C 转换器）。它还包含 isa.c 源文件，所有处理器模拟器都会使用该文件。
seq Source code for the SEQ and SEQ+ simulators. Contains the HCL file for homework problem 4.52. See file README for instructions on compiling the different versions of the simulator.
seq seq 和 seq+模拟器的源代码。包含作业问题 4.52 的 HCL 文件。有关编译不同版本模拟器的说明，请参阅文件 README。
pipe Source code for the PIPE simulator. Contains the HCL files for homework problems 4.54–4.58. See file README for instructions on compiling the different versions of the simulator.
pipe pipe 模拟器的源代码。包含作业问题 4.54－4.58 的 HCL 文件。有关编译不同版本模拟器的说明，请参阅文件 README。
y86-code Y86-64 assembly code for many of the example programs shown in the chapter. You can automatically test your modified simulators on these benchmark programs. See file README for instructions on how to run these tests. As a running example, we will use the program asum.ys in this subdirectory. This program is shown as CS:APP3e Figure 4.7. The compiled version of the program is shown in Figure 1.
y86-code 本章所示的许多示例程序的 y86-64 汇编代码。您可以在这些基准程序上自动测试修改后的模拟器。有关如何运行这些测试的说明，请参阅文件 README。作为一个运行示例，我们将使用该子目录中的 asum.ys 程序。该程序如 CS:APP3e 图 4.7 所示。程序的编译版本如图 1 所示。
ptest Scripts that generate systematic regression tests of the different instructions, the different jump possibilities, and different hazard possibilities. These scripts are very good at finding bugs in your homework solutions. See file README for instructions on how to run these tests.
Ptest 生成不同指令、不同跳转可能性和不同冒险可能性的系统回归测试脚本。这些脚本非常善于发现作业解决方案中的错误。有关如何运行这些测试的说明，请参阅文件 README。

```
1                                       | # Execution begins at address 0
2 0x000:                                |     .pos 0
```

```
3 0x000: 30f40002000000000000  |    irmovq stack, %rsp    # Set up stack pointer
4 0x00a: 803800000000000000    |    call main             # Execute main program
5 0x013: 00                    |    halt                  # Terminate program
6                              |
7                              | # Array of 4 elements
8 0x018:                       |    .align 8
9 0x018: 0d000d000d000000      | array:    .quad 0x000d000d000d
10 0x020: c000c000c0000000     |    .quad 0x00c000c000c0
11 0x028: 000b000b000b0000     |    .quad 0x0b000b000b00
12 0x030: 00a000a000a00000     |    .quad 0xa000a000a000
13                             |
14 0x038: 30f71800000000000000 | main:    irmovq array,%rdi
15 0x042: 30f60400000000000000 |    irmovq $4,%rsi
16 0x04c: 805600000000000000   |    call sum              # sum(array, 4)
17 0x055: 90                   |    ret
18                             |
19                             | # long sum(long *start, long count)
20                             | # start in %rdi, count in %rsi
21 0x056: 30f808000000000000000 | sum:          irmovq $8,%r8      # Constant 8
22 0x060: 30f901000000000000000 |    irmovq $1,%r9 # Constant 1
23 0x06a: 6300                 |    xorq %rax,%rax        # sum = 0
24 0x06c: 6266                 |    andq %rsi,%rsi        # Set CC
25 0x06e: 708700000000000000   |    jmp test              # Goto test
26 0x077: 50a70000000000000000 | loop:    mrmovq (%rdi),%r10    # Get *start
27 0x081: 60a0                 |    addq %r10,%rax        # Add to sum
28 0x083: 6087                 |    addq %r8,%rdi         # start++
29 0x085: 6196                 |    subq %r9,%rsi         # count--. Set CC
30 0x087: 747000000000000000   | test:    jne loop        # Stop when 0 #
31 0x090: 90                   |    ret                   # Return
32                             |
33                             | # Stack starts here and grows to lower addresses
34 0x200:                      |    .pos 0x200
35 0x200:                      | stack:
```

Figure 1: Sample object code file. This code is in the file asum.yo in the y86-code subdirectory.

图 1：示例目标代码文件。此代码位于 y86-code 子目录下的文件 asum.yo 中。

2 Utility Programs

2 实用程序

Once installation is complete, the misc directory contains two useful programs:

安装完成后，misc 目录包含两个有用的程序：

YAS The Y86-64 assembler. This takes a Y86-64 assembly code file with extension .ys and generates a file with extension .yo. The generated file contains an ASCII version of the object code, such as that shown in Figure 1 (The same program as shown in CS:APP3e Figure 4.8 with slightly different formatting.) The easiest way to invoke the assembler is to use or create assembly code files in the y86-code subdirectory. For example, to assemble the program in file asum.ys in this directory, we

use the command:
YAS Y86-64 汇编程序。这需要一个扩展名为.ys 的 Y86-64 汇编代码文件，并生成一个扩展名为.yo 的文件。生成的文件包含目标代码的 ASCII 版本，如图 1 所示（CS:APP3e 图 4.8 中显示的相同程序的格式略有不同。）调用汇编程序的最简单方法是在 y86-code 子目录中使用或创建汇编代码文件。例如，要汇编该目录下的文件 asum.ys 程序，我们使用以下命令：
linux> make asum.yo
YIS The Y86-64 instruction simulator. This program executes the instructions in a Y86-64 machine-level program according to the instruction set definition. For example, suppose you want to run the program asum.yo from within the subdirectory y86-code. Simply run:
YIS Y86-64 指令模拟器。该程序根据指令集定义执行 Y86-64 机器级程序中的指令。例如，假设您想运行 y86-code 子目录内的程序 asum.yo，只需运行：
linux> ../misc/yis asum.yo
YIS simulates the execution of the program and then prints changes to any registers or memory locations on the terminal, as described in CS:APP3e Section 4.1.
YIS 模拟程序的执行，然后在终端上打印任何寄存器或内存位置的更改，如 CS:APP3e 第 4.1 节所述。

3 Processor Simulators
3 处理器模拟器
For the three processors, SEQ, SEQ+, and PIPE, we have provided simulators SSIM, SSIM+, and PSIM respectively. Each simulator can be run in TTY or GUI mode:
对于 SEQ、SEQ+和 PIPE 这三个处理器，我们分别提供了模拟器 SSIM、SSIM+和 PSIM。每个模拟器可以在 TTY 或 GUI 模式下运行：
TTY mode Uses a minimalist, terminal-oriented interface. Prints everything on the terminal output. Not very convenient for debugging but can be installed on any system and can be used for automated testing. The default mode for all simulators.
TTY 模式使用一个极简的、面向终端的界面。终端输出上打印所有内容。调试不是很方便，但可以安装在任何系统上，并且可以用于自动化测试。所有模拟器的默认模式。
GUI mode Has a graphic user interface, to be described shortly. Very helpful for visualizing the processor activity and for debugging modified versions of the design. Requires installation of Tcl/Tk on your system. Invoked with the -g command line option. Running in GUI mode is only possible from within the directory (pipe or seq) in which the executable simulator program is located.
GUI 模式有一个图形用户界面，稍后介绍。对于可视化处理器活动和调试设计的修改版本非常有用。需要在系统上安装 Tcl/Tk。使用-g 命令行选项调用。只能从可执行模拟器程序所在的目录（流水线或顺序程序）才能以 GUI 模式运行。
3.1 Simulator Command Line Options
3.1 模拟器命令行选项
For all three simulators, you can specify several options from the command line:
对于所有三个模拟器，您可以从命令行指定几个选项：
-h Prints a summary of all of the command line options.
-h 打印所有命令行选项的摘要。
-g Run the simulator in GUI mode (the default is TTY mode).
-g 在 GUI 模式下运行模拟器（默认为 TTY 模式）。
-t (TTY mode only) Runs both the processor and the ISA simulators, comparing the resulting values

of the memory, register file, and condition codes. If no discrepancies are found, it prints the message “ISA Check Succeeds.” Otherwise, it prints information about the words of the register file or memory that differ. This feature is very useful for testing the processor designs.
-t（仅 TTY 模式）运行处理器和 ISA 模拟器，比较内存、寄存器文件和条件代码的结果值。如果没有发现差异，它将打印消息“ISA 检查成功”。否则，它将输出有关不同的寄存器文件或内存字的信息。此功能对于测试处理器设计非常有用。
-l m (TTY mode only) Sets the instruction limit, executing at most m instructions before halting (the default limit is 10000 instructions).
-l m（仅限 TTY 模式）设置指令限制，在停机前最多执行 m 条指令（默认限制为 10000 条指令）。
-v n (TTY mode only) Sets the verbosity level to n, which must be between 0 and 2 with a default value of 2. Simulators running in GUI mode must be invoked with the name of an object file on the command line. In TTY mode, the object file name is optional, coming from stdin by default. Here are some typical invocations of the simulators from within the seq subdirectory:
-v n（仅限 TTY 模式）将详细级别设置为 n，该级别必须介于 0 和 2 之间，默认值为 2。在 GUI 模式下运行的模拟器必须使用命令行上的目标文件名调用。在 TTY 模式下，目标文件名是可选的，默认来自标准输入。以下是 seq 子目录中模拟器的一些典型调用方法：

```
linux> ./ssim -h
linux> ./ssim+ -t < ../y86-code/asum.yo
linux> ./ssim -g ../y86-code/asum.yo
```

The first case prints a summary of the command line options for SSIM. The second case runs SSIM+ in TTY mode, reading object file asum.yo from stdin. The resulting register and memory values are compared with those from the higher-level ISA simulator. The third case runs SSIM in GUI mode, executing the instructions in object code file asum.yo from the y86-code subdirectory. The same invocations would work for the PIPE simulator PSIM from within the pipe subdirectory.
第一种情况打印 SSIM 的命令行选项摘要。第二种情况以 TTY 模式运行 SSIM+，从标准输入读目标文件 asum.yo。寄存器中的结果和内存值与来自高级 ISA 模拟器的值进行比较。第三种情况在 GUI 模式下运行 SSIM，执行 y86-code 子目录中目标代码文件 asum.yo 中的指令。同样的调用也适用于 pipe 子目录中的 PIPE 模拟器 PSIM。

3.2 GUI Version of SEQ Simulator
3.2 SEQ 模拟器的 GUI 版本

The GUI version of the SEQ processor simulator is invoked from within the seq subdirectory with an object code filename on the command line:
SEQ 处理器模拟器的 GUI 版本从 SEQ 子目录中调用，命令行上指定一个目标代码文件名：

```
linux> ./ssim -g ../y86-code/asum.yo &
```

where the “&” at the end of the command line allows the simulator to run in background mode. The simulation program starts up and creates three windows, as illustrated in Figures 2–4.
其中，命令行末尾的“&”允许模拟器在后台模式下运行。模拟程序启动并创建三个窗口，如图 2-4 所示。

The first window (Figure 2) is the main control panel. If the HCL file was compiled by HCL2C with the -n name option, then the title of the main control window will appear as “Y86-64 Processor: name” Otherwise it will appear as simply “Y86-64 Processor.”
第一个窗口（图 2）是主控制面板。如果 HCL 文件是由 HCL2C 使用-n name 选项编译的，则主控制窗口的标题将显示为“Y86-64 处理器：名称”，否则它将显示为简单的“Y86-64 处理

器”

The main control window contains buttons to control the simulator as well as status information about the state of the processor. The different parts of the window are labeled in the figure:

主控制窗口包含用于控制模拟器的按钮以及有关处理器状态的状态信息。图中标记了窗口的不同部分：

Control: The buttons along the top control the simulator. Clicking the Quit button causes the simulator to exit. Clicking the Go button causes the simulator to start running. Clicking the Stop button causes the simulator to stop temporarily. Clicking the Step button causes the simulator to execute one instruction and then stop. Clicking the Reset button causes the simulator to return to its initial state, with the program counter at address 0, the registers set to zeros, the memory erased except for the program, the condition codes set with ZF = 1, CF = 0, and OF = 0, and the program status set to AOK. The slider below the buttons controls the speed of the simulator when it is running. Moving it to the right makes the simulator run faster.

控制：顶部的按钮控制模拟器。单击 Quit（退出）按钮会导致模拟器退出。单击 Go 按钮可使模拟器开始运行。单击 Stop（停止）按钮可使模拟器暂时停止。单击 Step（单步）按钮可使模拟器执行一条指令，然后停止。点击 Reset（复位）按钮使模拟器返回其初始状态，此时，程序计数器位于地址 0，寄存器设置为 0，内存除了程序外都清零，条件码设置为 ZF= 1，CF = 0 而且 OF = 0，程序状态设置为 AOK。按钮下面的滑动条控制模拟器运行时的速度，向右滑动会使模拟器运行的更快。

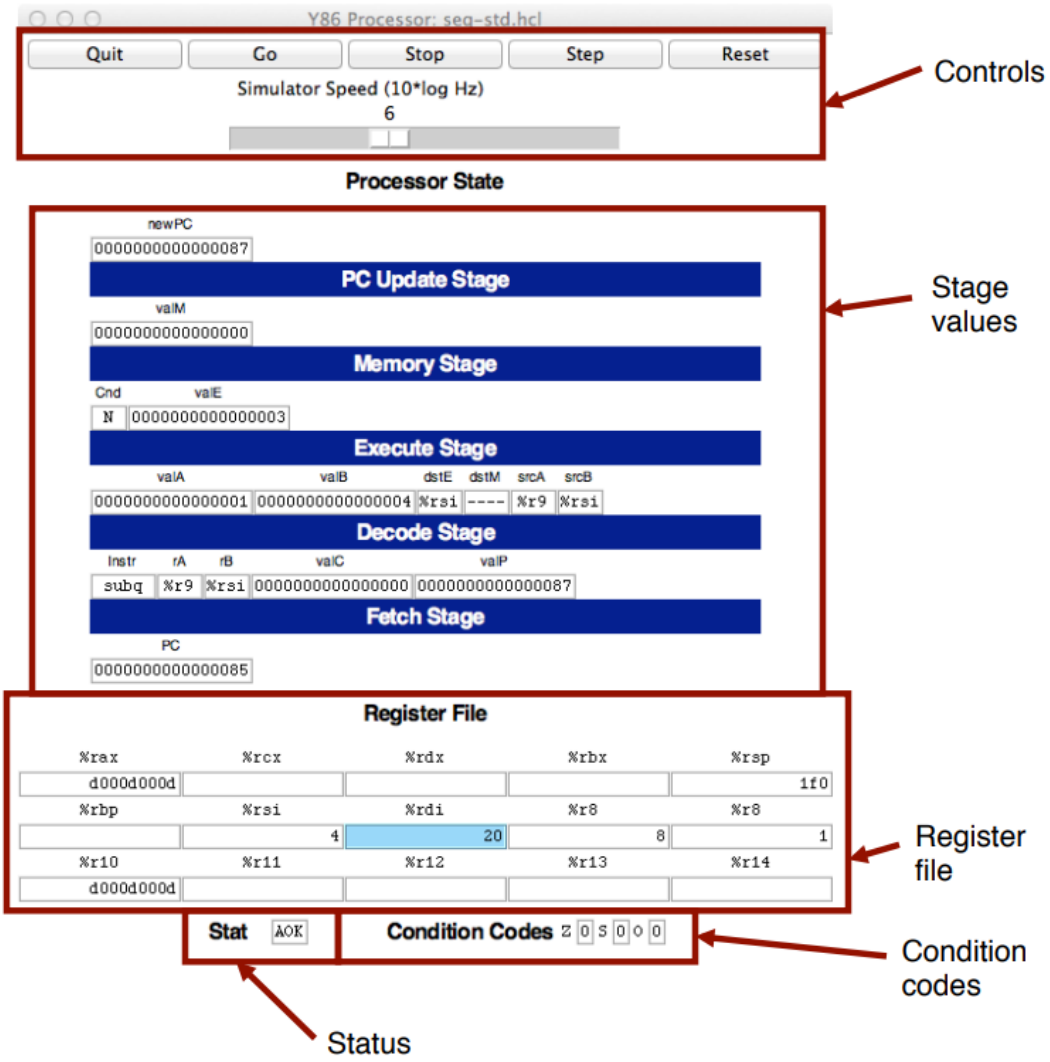


Figure 2: Main control panel for SEQ simulator

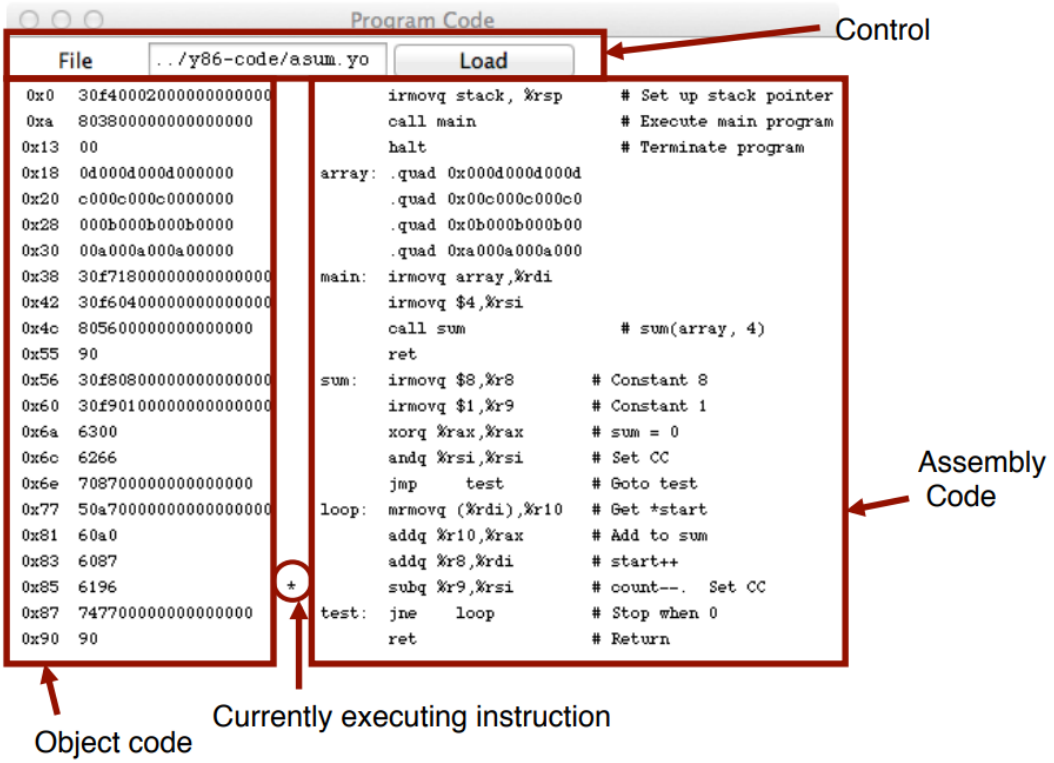


Figure 3: Code display window for SEQ simulator

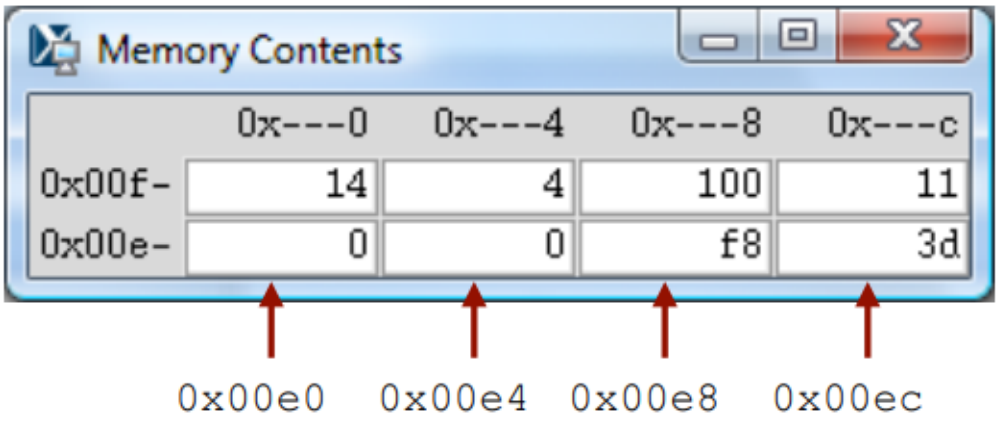


Figure 4: Memory display window for SEQ simulator

Stage values: This part of the display shows the values of the different processor signals during the current instruction evaluation. These signals are almost identical to those shown in CS:APP3e Figure 4.23. The main difference is that the simulator displays the name of the instruction in a field labeled Instr, rather than the numeric values of icode and ifun. Similarly, all register identifiers are shown using their names, rather than their numeric values, with "----" indicating that no register access is required.

流水线阶段值：此部分显示当前指令评估期间不同处理器信号的值。这些信号几乎与 CS:APP3e 图 4.23 中显示的信号相同。主要区别在于模拟器在标记为 Instr 的字段中显示指令的名称，而不是 icode 和 ifun 的数值。类似地，所有寄存器标识符都使用其名称而不是数字值显示，其中"----"表示不需要访问寄存器。

Register file: This section displays the values of the 15 program registers. The register that has been

updated most recently is shown highlighted in light blue. Register contents are not displayed until after the first time they are set to nonzero values. Remember that when an instruction writes to a program register, the register file is not updated until the beginning of the next clock cycle. This means that you must step the simulator one more time to see the update take place.
寄存器文件：此部分显示 15 个程序寄存器的值。最近更新的寄存器以浅蓝色突出显示。寄存器内容在第一次设置为非零值后才会显示。请记住，当指令写入程序寄存器时，直到下一个时钟周期开始，寄存器文件才会更新。这意味着您必须再次单步执行模拟器才能看到更新的发生。
Stat: This shows the status of the current instruction being executed. The possible values are:
Stat：显示当前正在执行的指令的状态。可能的值为：
AOK: No problem encountered.
AOK：没有遇到问题。
ADR: An addressing error has occurred either trying to read an instruction or trying to read or write data. Addresses cannot exceed 0x0FFF.
ADR：试图读取指令或试图读取或写入数据时发生寻址错误。地址不能超过 0x0FFF。
INS: An illegal instruction was encountered.
INS：遇到非法指令。
HLT: A halt instruction was encountered.
HLT:遇到停机指令。
Condition codes: These show the values of the three condition codes: ZF, SF, and OF.
条件码：这些代码显示了三个条件代码的值：ZF、SF 和 OF。
Remember that when an instruction changes the condition codes, the condition code register is not updated until the beginning of the next clock cycle. This means that you must step the simulator one more time to see the update take place.
请记住，当指令更改条件码时，直到下一个时钟周期开始，条件码寄存器才会更新。这意味着您必须再次单步执行模拟器才能看到更新的发生。
The processor state illustrated in Figure 2 is for the first execution of line 29 of the asum.yo program shown in Figure 1. We can see that the program counter is at 0x085, that it has processed the instruction addq %r8, %rdi, that register %rax holds 0xd000d000d, the sum of the first array element, and %rsi holds 4, the count that is about to be decremented. Register %rdi holds 0x020, the address of the second array element. There is a pending write of 0x03 to register %rsi (since dstE is set to %rsi and valE is set to 0x03). This write will take place at the start of the next clock cycle.
第一次执行 asum.yo 程序的第 29 行时处理器状态如图 2 所示，asum.yo 程序如图 1 所示。我们可以看到，程序计数器位于 0x085，此时它已处理了指令 addq %r8，%rdi，寄存器%rax 保存 0xd000d000d，第一个数组元素的总和，而且%rsi 保存 4，即将减少的计数。寄存器%rdi 保存 0x020，即第二个数组元素的地址。有一个挂起的 0x03 写入寄存器%rsi（因为 dstE 设置为%rsi，而 valE 设置为 0x03）。此写入将在下一个时钟周期开始时进行。
The window depicted in Figure 3 shows the object code file that is being executed by the simulator. The edit box identifies the file name of the program being executed. You can edit the file name in this window and click the Load button to load a new program. The left hand side of the display shows the object code being executed, while the right hand side shows the text from the assembly code file. The center has an asterisk (*) to indicate which instruction is currently being simulated. This corresponds to line 29 of the asum.yo program shown in Figure 1.

图 3 所示的窗口显示了模拟器正在执行的目标代码文件。编辑框标识正在执行的程序的文件名。您可以在此窗口中编辑文件名，然后单击 Load（加载）按钮加载新程序。显示屏的左侧显示正在执行的目标代码，而右侧显示汇编代码文件的文本。中心有一个星号（*），表示当前正在模拟的指令。这对应于图 1 所示 asum.yo 程序的第 29 行。

The window shown in Figure 4 shows the contents of the memory. It shows only those locations between the minimum and maximum addresses that have changed since the program began executing. Each row shows the contents of two memory words. Thus, each row shows 16 bytes of the memory, where the addresses of the bytes differ in only their least significant hexadecimal digits. To the left of the memory values is the “root” address, where the least significant digit is shown as “-”. Each column then corresponds to words with least significant address digits 0x0, and 0x8. The example shown in Figure 4 has arrows indicating memory locations 0x01f0 and 0x01f8.

图 4 所示的窗口显示了内存的内容。它只显示自程序开始执行以来更改的最小和最大地址之间的位置。每行显示两个内存字的内容。因此，每行显示 16 个字节的内存，其中字节的地址仅在最低有效十六进制数字上有所不同。内存值的左边是“根”地址，其中最低有效数字显示为“-”。然后，每一列对应于具有最低有效地址数字为 0x0 和 0x8 的字。图 4 所示的示例具有箭头，指示内存位置 0x01f0 和 0x01f8。

The memory contents illustrated in the figure show the stack contents of the asum.yo program shown in Figure 1 during the execution of the sum procedure. Looking at the stack operations that have taken place so far, we see that %rsp was initialized to 0x200 (line 3). The call to main on line 4 pushes the return pointer 0x013, which is written to address 0x01f8. Procedure main calls sum (line 16), causing the return pointer 0x055 to be written to address 0x01f0. That accounts for all of the words shown in this memory display, and for the stack pointer being set to 0x01f0.

图中所示的内存内容显示了图 1 所示 asum.yo 程序在执行求和过程期间栈内容。查看到目前为止发生的栈操作，我们看到%rsp 被初始化为 0x200（第 3 行）。第 4 行对 main 的调用将返回指针 0x013 压栈，返回指针被写入地址 0x01f8。过程 main 调用 sum（第 16 行），导致返回指针 0x055 被写入地址 x01f0。这说明了内存窗口中显示的所有字，以及栈指针被设置为 0x01f0 的原因。

3.3 PIPE Simulator

3.3 PIPE 模拟器

The PIPE simulator also generates three windows. Figure 5 shows the control panel. It has the same set of controls, and the same display of the register file, status, and condition codes. The middle section shows the state of the pipeline registers. The different fields correspond to those in CS:APP3e Figure 4.52. At the bottom of this panel is a display showing the number of cycles that have been simulated (not including the initial cycles required to get the pipeline flowing), the number of instructions that have completed, and the resulting CPI.

PIPE 模拟器也生成三个窗口。图 5 显示了控制面板。它具有相同的控件集，以及相同的寄存器文件、状态和条件码显示。中间部分显示流水线寄存器的状态。不同的字段对应于 CS:APP3e 图 4.52 中的字段。该面板底部显示了已模拟的周期数（不包括使流水线流动所需的初始周期）、已完成的指令数以及生成的 CPI。

As illustrated in the close-up view of Figure 6, each pipeline register is displayed with two parts. The upper values in white boxes show the current values in the pipeline register. The lower values with a gray background show the inputs to pipeline register. These will be loaded into the register on the next clock cycle, unless the register bubbles or stalls.

如图 6 的特写视图所示，每个流水线寄存器都由两部分显示。白色框中的上面值显示流水线

寄存器中的当前值。下面的值以灰色背景显示流水线寄存器的输入。这些将在下一个时钟周期加载到寄存器中，除非寄存器插入气泡或暂停。

The flow of values through the PIPE simulator is quite different from that for the SEQ simulator. With SEQ, the control panel shows the values resulting from executing a single instruction. Each step of the simulator performs one complete instruction execution. With PIPE, the control panel shows the values for the multiple instructions flowing through the pipeline. Each step of the simulator performs just one stage's worth of computation for each instruction.

通过 PIPE 模拟器的数据流与 SEQ 模拟器的数据流动大不相同。使用 SEQ，控制面板显示执行单个指令产生的值。模拟器的每个步骤执行一个完整的指令执行。使用 PIPE，控制面板显示流经流水线的多条指令的值。模拟器的每个步骤对每个指令只执行一个阶段的计算。

Figure 7 shows the code display for the PIPE simulator. The format is similar to that for SEQ, except that rather than a single marker indicating which instruction is being executed, the display indicates which instructions are in each stage of the pipeline, using characters F, D, E, M, and W, for the fetch, decode, execute, memory, and write-back stages.

图 7 显示了 PIPE 模拟器的代码显示。该格式与 SEQ 的格式类似，不同之处在于，该显示器不是用一个标记来指示正在执行的指令，而是用字符 F、D、E、M 和 W 来指示流水线的每个阶段中有哪些指令，用于取指、译码、执行、内存和写回阶段。

The PIPE simulator also generates a window to display the memory contents. This has an identical format to the one shown for SEQ (Figure 4).

PIPE 模拟器也生成一个窗口来显示内存内容。其格式与 SEQ 所示格式相同（图 4）。

The example shown in Figures 5 and 7 show the status of the pipeline when executing the loop in lines 26–30 of Figure 1. We can see that the simulator has begun the first iteration of the loop, having entered the loop by jumping to the test portion (line 30). The status of the stages is as follows:

图 5 和图 7 所示的示例显示了在图 1 的第 26-30 行中执行循环时流水线的状态。我们可以看到，模拟器已经开始了循环的第一次迭代，通过跳到测试部分（第 30 行）进入了循环。各阶段的状态如下：

Write back: The jne instruction (line 30) of the initial test is finishing.

写回：初始测试的 jne 指令（第 30 行）正在完成。

Figure 5: Main control panel for PIPE simulator 10 Current state Register inputs

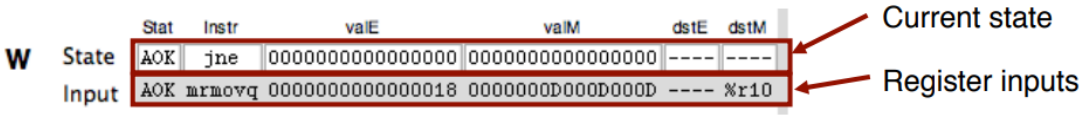


Figure 6: View of single pipe register in control panel for PIPE simulator

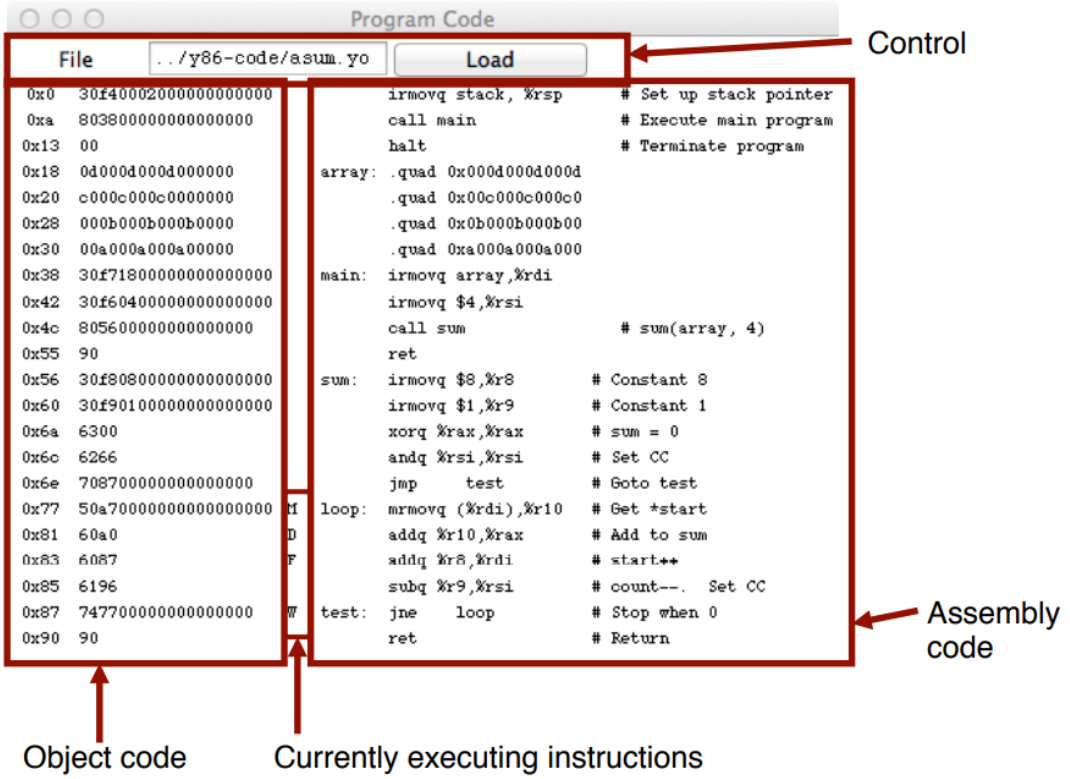


Figure 7: Code display window for PIPE simulator

Memory: The mrmovq instruction (line 26) has just read 0x0D000D000D from address 0x018. We can see the address in valE of pipeline register M, and the value read from memory at the input of valM to pipeline register W.

内存：mrmovq 指令（第 26 行）刚刚从地址 0x018 读取了 0x0D000D000D。我们可以看到地址在流水线寄存器 M 的字段 valE 中，以及在 valM 输入到流水线寄存器 W 时从内存读取的值。

Execute: This stage contains a bubble. The bubble was inserted due to the load-use dependency between the mrmovq instruction (line 26) and the addq instruction (line 27). It can be seen that this bubble acts like a nop instruction. This explains why there is no instruction in Figure 7 labeled with “E.”

执行：此阶段包含一个气泡。由于 mrmovq 指令（第 26 行）和 addq 指令（27 行）之间存在加载使用依赖关系，所以插入了气泡。可以看出，这个气泡的作用类似于 nop 指令。这解释了为什么图 7 中没有标有“E”的指令。

Decode: The addq instruction (line 27) has just read 0x0 from register %rax. It also read 0x00D from register %r10, but we can see that the forwarding logic has instead used the value 0x0D000D000D that has just been read from memory (seen as the input to valM in pipeline register W) as the new value of valA (seen as the input to valA in pipeline register E).

译码：addq 指令（第 27 行）刚刚从寄存器%rax 读取 0x0。它还从寄存器%r10 读取 0x00D，但我们可以看到，转发逻辑使用了刚刚从内存读取的值 0x0D000D000D（在流水线寄存器 W 中被视为 valM 的输入）作为 valA 的新值（在流水线寄存器 E 中被看作 valA 的输入）。

Fetch: An addq instruction (line 28) has just been fetched from address 0x083. The new value of the PC is predicted to be 0x085.

取指：刚刚从地址 0x083 提取了 addq 指令（第 28 行）。预计 PC 的新值为 0x085。

Associated with each stage is its status field Stat This field shows the status of the instruction in that stage of the pipeline. Status AOK means that no exception has been encountered. Status value BUB indicates that a bubble is in this stage, rather than a normal instruction. Other possible status values are: ADR when an invalid memory location is referenced, INS when an illegal instruction code is encountered, PIP when a problem arose in the pipeline (this occurs when both the stall and the bubble signals for some pipeline register are set to 1), and HLT when a halt instruction is encountered. The simulator will stop when any of these last four cases reaches the write-back stage.
与每个阶段关联的是其状态字段 Stat。此字段显示流水线该阶段中指令的状态。状态 AOK 表示未遇到异常。状态值 BUB 表示此阶段有气泡，而不是正常指令。其他可能的状态值包括：引用无效内存位置时的 ADR，遇到非法指令代码时的 INS，流水线中出现问题时的 PIP（当某些流水线寄存器的暂停信号和气泡信号都设置为 1 时发生），以及遇到停机指令时的 HLT。当最后四种情况中的任何一种达到回写阶段时，模拟器将停止。
Carrying the status for an individual instruction through the pipeline along with the rest of the information about that instruction enables precise handling of the different exception conditions, as described in CS:APP3e Section 4.5.6.
如 CS:APP3e 第 4.5.6 节所述，通过流水线携带单个指令的状态以及该指令的其余信息，可以精确处理不同的异常条件。
4 Some Advice
4 一些建议
The following are some miscellaneous tips, learned from experience we have gained in using these simulators.
以下是从我们使用这些模拟器的经验中学到的一些杂项提示。
• Get familiar with the simulator operation. Try running some of the example programs in the y86-code directory. Make sure you understand how each instruction gets processed for some small examples. Watch for interesting cases such as mispredicted branches, load interlocks, and procedure returns.
•熟悉模拟器操作。尝试运行 y86-code 目录中的一些示例程序。对于一些小示例，请确保您了解每条指令是如何处理的。注意有趣的情况，例如预测失误的分支、加载互锁和过程返回。
• You need to hunt around for information. Seeing the effect of data forwarding is especially tricky. There are seven possible sources for signal valA in pipeline register E, and six possible sources for signal valB. To see which one was selected, you need to compare the input to these pipeline register fields to the values of the possible sources. The possible sources are:
•你需要四处寻找信息。看到数据转发的效果尤其棘手。流水线寄存器 E 中信号 valA 有七个可能的来源，信号 valB 有六个可能的来源。要查看选择了哪一个，需要将这些流水线寄存器字段的输入与可能的来源的值进行比较。可能的来源包括：
R[d_srcA] The source register is identified by the input to srcA in pipeline register E. The register contents are shown at the bottom.
R[d_srcA]源寄存器由流水线寄存器 E 中 srcA 的输入标识。寄存器内容显示在底部。
R[d_srcB] The source register is identified by the input to srcB in pipeline register E. The register contents are shown at the bottom.
R[d_srcB]源寄存器由流水线寄存器 E 中 srcB 的输入标识。寄存器内容显示在底部。
D_valP This value is part of the state of pipeline register D.
D_valP 此值是流水线寄存器 D 状态的一部分。
e_valE This value is at the input to field valE in pipeline register M.

e_valE 该值位于流水线寄存器 M 中字段 valE 的输入端。
M_valE This value is part of the state of pipeline register M.
M_valE 此值是流水线寄存器 M 状态的一部分。
m_valM This value is at the input to field valM in pipeline register W.
m_valM 此值位于流水线寄存器 W 中字段 valM 的输入处。
W_valE This value is part of the state of pipeline register W.
W_valE 此值是流水线寄存器 W 状态的一部分。
W_valM This value is part of the state of pipeline register M.
W_valM 此值是流水线寄存器 M 状态的一部分。
• Do not overwrite your code. Since the data and code share the same address space, it is easy to have a program overwrite some of the code, causing complete chaos when it attempts to execute the overwritten instructions. It is important to set up the stack to be far enough away from the code to avoid this.
•不要覆盖代码。由于数据和代码共享相同的地址空间，程序很容易覆盖某些代码，当它试图执行被覆盖的指令时，会造成完全混乱。将栈设置得离代码足够远以避免这种情况，这一点很重要。
• Avoid large address values. The simulators do not allow any addresses greater than 0x0FFF. In addition, the memory display becomes unwieldy if you modify memory locations spanning a wide range of addresses.
•避免地址值过大。模拟器不允许任何大于 0x0FFF 的地址。此外，如果修改跨越广泛地址范围的内存位置，内存显示将变得不方便。
• Be aware of some “features” of the GUI-mode simulators (SSIM, SSIM+, and PSIM.)
•了解 GUI 模式模拟器的一些“功能”（SSIM、SSIM+和 PSIM）
– You must must execute the programs from their home directories. In other words, to run SSIM or SSIM+, you must be in the seq directory, while you must be in the pipe subdirectory to run PSIM. This requirement arises due to the way the Tcl interpreter locates the configuration file for the simulator.
–必须从主目录执行程序。换句话说，要运行 SSIM 或 SSIM+，必须位于 seq 目录中，而要运行 PSIM，则必须位于 pipe 子目录中。这个需求是由于 Tcl 解释器定位模拟器配置文件的方式而产生的。
– If you are running in GUI mode on a Unix box, remember to initialize the DISPLAY environment variable:
–如果在 Unix 机器上以 GUI 模式运行，请记住初始化 DISPLAY 环境变量：
unix> setenv DISPLAY myhost.edu:0
– With some Unix X Window managers, the “Program Code” window begins life as a closed icon. If you don’t see this window when the simulator starts, you’ll need to expand the expand manually by clicking on it.
–对于某些 Unix X Window 管理器，“程序代码”窗口以关闭图标开始。如果模拟器启动时没有看到此窗口，则需要通过单击 expand 手动展开。
– With some Microsoft Windows X servers, the “Memory Contents” window does not automatically resize itself when the memory contents change. In these cases, you’ll need to resize the window manually to see the memory contents.
–对于某些 Microsoft Windows X 服务器，当内存内容更改时，“内存内容”窗口不会自动调整大小。在这些情况下，您需要手动调整窗口大小以查看内存内容。

– The simulators will terminate with a segmentation fault if you ask them to execute a file that is not a valid Y86-64 object file.

– 如果您要求模拟器执行一个不是有效 Y86-64 目标文件，模拟器将因段错误而终止。