Лабораторная работа №3

Эмулятор RISC-V

Инструментарий и требования к работе

Допустимые языки	C	C++	Python (только НоД)
Стандарты / версии	C23	C++23	3.13
Требования для всех работ	Правила оформления и написания работ		
Материалы	riscv-asm-manual riscv-isa-release-2025-10-16 **unprivileged**		

Описание работы

Необходимо смоделировать работу системы “процессор-кэш-память” при выполнении кода на RISC-V с политиками вытеснения LRU и bit-pLRU.

Аргументы программе передаются через командную строку и могут располагаться в любом порядке относительно друг друга.

Аргументы	**Комментарии**
`-i <имя_файла>`	Состояние регистров и оперативной памяти регистров перед началом исполнения.
`-o <имя_файла> <адрес_начала> <размер>`	[Опциональный параметр] Состояние регистров и оперативной памяти по окончании исполнения (как её видит исполняемый код).

Обратите внимание: `<адрес_начала>` и `<размер>` могут быть указаны как в десятичной системе, так и в шестнадцатиричной (с префиксом 0x).

Вывод результата моделирования (число попаданий к общему числу обращений в процентах) производится в поток вывода в формате `printf`.

Кроме общего процента попадания необходимо вычислить процент попадания отдельно для инструкций и для данных.

```
"| replacement | hit_rate | instr_hit_rate | data_hit_rate |
instr_access |  instr_hit   | data_access  |   data_hit   |\n"
           "| :---------- | :------: | ------------: |
------------: | -----------: | -----------: | -----------: |
-----------: |\n"
           "| LRU         | %3.4f%% |       %3.4f%% |      %3.4f%%
| %12d | %12d | %12d | %12d |\n"
           "| bpLRU       | %3.4f%% |       %3.4f%% |      %3.4f%%
| %12d | %12d | %12d | %12d |\n"
```

В случае, когда не было обращений к памяти, следует выводить `nan%` и `0`. Для вывода результата настоятельно рекомендуется использовать `printf` в C и `printf/format` в C++.

Если какая-то из политик вытеснения не реализована, то выводится `-`. Например, если нет реализации pLRU, то вывод про него будет:

```
"| bpLRU       |       -  |             - |             - |
- |           - |           - |           - |\n"
```

Расхождение даже в один пробельный символ или потерянный знак `%` приведёт к тому, что ответ не будет зачтён.

Входной файл

Двоичный файл в формате:

Первые 32*4 байта	состояние регистров: `pc, x1, ..., x31`
Фрагменты оперативной памяти в формате	4 байта – адрес начала 4 байта – размер данного фрагмента N N байт – фрагмент оперативной памяти

Состояние памяти вне поданных фрагментов неопределено.

Необходимо поддерживать команды RV32I и RV32M.

Выходной файл

Аналогичен входному, но содержит только 1 фрагмент памяти. Состояние памяти с точки зрения ISA, другими словами – состояние памяти, которое видит исполняемая программа.

Исполнение команд

Команды исполняются последовательно и читаются из памяти только перед непосредственным исполнением одной операцией чтения.

Концом исполнения кода программы считается переход на адрес, содержащийся в регистре `ra` на *старте* программы.

Гарантируется, что в памяти команды и данные не пересекаются (например, команда записи в память не изменит исходные команды).

Многобайтовое обращение считается за одну операцию. Гарантируется, что все обращения к памяти выровненные – адрес начала кратен размеру порции данных.

Команды `ecall` и `ebreak` приводят к завершению исполнения.

Кэш-память

Моделируемый кэш **общий для данных и команд**. По реализации должно быть однозначно понятно, что кэш реализован нужной конфигурации!

Начальное состояние: кэш пуст, все кэш-линии в состоянии invalid.

Все приведённые ниже переменные должны в явном виде располагаться в коде (желательно в одном файле):

Переменная/константа	**Пояснение**

MEMORY_SIZE	размер памяти (в байтах)
CACHE_SIZE	размер кэша без учёта служебной информации (в байтах)
CACHE_LINE_SIZE	размер кэш-линии (в байтах)
CACHE_LINE_COUNT	кол-во кэш-линий
CACHE_WAY	ассоциативность
CACHE_SET_COUNT	кол-во блоков кэш-линий
ADDRESS_LEN	длина адреса (в битах)
CACHE_TAG_LEN	длина тэга адреса (в битах)
CACHE_INDEX_LEN	длина индекса блока кэш-линий (в битах)
CACHE_OFFSET_LEN	длина смещения внутри кэш-линии (в битах)

Интерпретация адреса кэшем (слева старшие биты, справа – младшие):

tag	**index**	**offset**
CACHE_TAG_LEN	CACHE_INDEX_LEN	CACHE_OFFSET_LEN

Параметры системы

Ниже указана конфигурация. Пустые ячейки – параметр необходимо вычислить самостоятельно.

Параметры	**Значения**
Конфигурация кэша	look-through write-back write-allocate
Политика вытеснения кэша	LRU и bit-pLRU
MEMORY_SIZE	256 Кбайт
ADDRESS_LEN	
CACHE_TAG_LEN	8
CACHE_INDEX_LEN	
CACHE_OFFSET_LEN	

CACHE_SIZE	
CACHE_LINE_SIZE	
CACHE_LINE_COUNT	
CACHE_SET_COUNT	32
CACHE_WAY	4

task.bin

Помимо программы в репозиторий необходимо загрузить `task.bin`, который будет содержать такое состояние памяти и регистров, при котором в результате исполнения:

1. произойдёт хотя бы одно вытеснение из кэша;
2. для хотя бы одной из политик вытеснения будут достигнуты проценты (проценты представлены в таблице курса на листе лабораторной).

Будет полезным приложить в репозиторий task.asm, в котором вы опишите на ассемблере RISC-V тот код, который лежит в вашем task.bin. Так будет проще дать вам информативную обратную связь при возникновении проблем.

В автотестах будет производиться запуск с вашим `task.bin` файлом.

Полезное для понимания: симулятор RISC-V

GitHub - mortbopet/Ripes: A graphical processor simulator and assembly editor for the RISC-V ISA

Визуальный симулятор архитектуры и редактор ассемблерного кода, созданный для RISC-V.

Вы можете загрузить туда код на C/C++ и при помощи gnu toolchain скопилировать его или сразу загрузить elf / двоичный сырой файл.

Загруженный код можно исполнить. На выбор доступны разные конфигурации конвейера и кэша.