Лабораторная работа №4

Кэш и кодирование команд

Инструментарий и требования к работе

Допустимые языки	C++	Python	Java
Стандарты / версии	C++20	3.12.5	Temurin-21.0.4
Требования для всех работ	Правила оформления и написания работ		
Чтобы работа была принята на проверку, необходимо выполнить расчёт параметров моделируемой системы и написать программу, выполняющую моделирование с политикой LRU.			

Описание работы

Необходимо смоделировать работу процессора и кэша при выполнении кода на RISC-V с политиками вытеснения LRU и bit-pLRU. Дополнительно нужно реализовать перевод входного asm-кода в машинный код.

Аргументы программе передаются через командную строку. Набор возможных аргументов приведён в таблице ниже. Аргументы могут располагаться в любом порядке относительно друг друга.

Аргументы	**Комментарии**
`--asm <имя_файла_с_кодом_ассемблера>`	Параметр, через который передаётся файл с кодом на ассемблере, который необходимо исполнить.
`--bin <имя_файла_с_машинным_кодом>`	Параметр, указывающий имя файла, в который нужно записать результат перевода ассемблера в машинный код. Если работа выполняется без этой части, то в таком случае необходимо написать в поток

	вывода ошибок `'Compiling asm code is not supported'` и завершиться с нулевым кодом возврата.

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

```
LRU\thit rate: %3.5f%%\npLRU\thit rate: %3.5f%%\n
```

В случае, когда не было обращений к памяти, следует выводить `nan%`

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

Для вывода результата настоятельно рекомендуется использовать printf в C и printf/format в C++ и аналогичные конструкции других языков. Использовать другие варианты не запрещается, но результат должен быть эквивалентным. Расхождение даже в один пробельный символ или потерянный знак процента приведёт к неправильному ответу.

Кодирование команд

Программа должна принимать на вход файл с кодом ассемблера RISC-V и уметь его исполнять. Архитектура набора команд RISC-V (**unprivileged** из riscv-isa-release-b796659-2024-11-14).

Формат регистров: ABI. Псевдонимы команд не нужно поддерживать.

Должен поддерживаться следующий набор команд RISC-V: RV32I, RV32M. Расширения (Zifence, Zicsr) поддерживать не нужно.

Константы в командах необходимо обрабатывать как десятичные, так и шестнадцатеричные (начинаются с префикса **0x**, буквы в любом регистре).

Аргументы в командах следуют через `','` . Перед командой, между аргументами или в конце возможно произвольное количество пробельных символов (пробел, табуляция, перевод строки).

Команды и названия регистров в нижнем регистре (строчными буквами).

Пример asm-кода RISC-V

```
addi s0, zero, 0
addi s1, zero, 0
addi a0, zero,   30

beq  a0,  zero, 28
andi s2, a0, 1
srli a0, a0, 1

beq s2 , zero, 0x4
addi s0, s1, 0
addi s1, s1, 1

  jal zero, -24
```

В файле могут быть пустые строки, как показано в примере выше.

В дополнение к вышеописанному необходимо написать функцию/набор функций/класс, которые преобразуют код из входного файла в машинный код. Результат записывается в файл, имя которого передано в аргументах командной строки, в двоичном формате.

Пример перевода кода RISC-V в машинный код

Пример кода на ассемблере	**Пример машинного кода (значения байт в шестнадцатеричной системе)**
`addi a5, a5, 400`	`93 87 07 19`
`jal zero, 0x9C`	`6f 00 c0 09`
`sw a5, -36, s0`	`23 2e f4 fc`
`bge a5, a4, -88`	`e3 d4 e7 fa`

Во всех указанных примерах размер выходного файла – 4 байта.

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, или переход на адрес за пределами поданных команд.

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

Команды располагаются в памяти начиная с адреса **`0x10000`**.

Гарантируется, что последовательность выполнения команд (например, число итераций цикла) и адреса памяти, к которым они обращаются, не зависит от данных, хранящихся в оперативной памяти.

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

Команды **`ecall`** и **`ebreak`** интерпретировать как **`nop`**.

Кэш-память

Моделируемый кэш **общий для данных и команд**.

Начальное состояние – кэш пуст (все кэш-линии в состоянии invalid).

Переменные/константы (как они должны называться в коде):

- `MEM_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_SETS – кол-во блоков кэш-линий
- ADDR_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

В отчете нужно подробно описать устройство кэш-линии: как хранятся данные, как хранится и сколько занимает служебная информация.

Параметры системы

Везде, где не указаны значения, нужно самостоятельно вычислить. Результат расчёта должен быть описан в отчёте со всеми формулами и пояснениями.

Ниже указаны различные варианты, варианты выдаются через таблицу курса. Все расчёты в миниотчёте приводятся только для вашего варианта.

Параметры (вариант 1)

MEM_SIZE	вычислить самостоятельно
ADDR_LEN	18 бит
Конфигурация кэша	look-through write-back
Политика вытеснения кэша	LRU и bit-pLRU
CACHE_WAY	вычислить самостоятельно
CACHE_TAG_LEN	вычислить самостоятельно
CACHE_INDEX_LEN	3 бита

CACHE_OFFSET_LEN	вычислить самостоятельно
CACHE_SIZE	вычислить самостоятельно
CACHE_LINE_SIZE	64 байт
CACHE_LINE_COUNT	32
CACHE_SETS	вычислить самостоятельно

Параметры (вариант 2)

MEM_SIZE	вычислить самостоятельно
ADDR_LEN	18 бит
Конфигурация кэша	look-through write-back
Политика вытеснения кэша	LRU и bit-pLRU
CACHE_WAY	4
CACHE_TAG_LEN	вычислить самостоятельно
CACHE_INDEX_LEN	вычислить самостоятельно
CACHE_OFFSET_LEN	вычислить самостоятельно
CACHE_SIZE	вычислить самостоятельно
CACHE_LINE_SIZE	32 байта
CACHE_LINE_COUNT	вычислить самостоятельно
CACHE_SETS	32

Параметры (вариант 3)

MEM_SIZE	512 Кбайт
ADDR_LEN	вычислить самостоятельно
Конфигурация кэша	look-through write-back
Политика вытеснения кэша	LRU и bit-pLRU
CACHE_WAY	вычислить самостоятельно
CACHE_TAG_LEN	вычислить самостоятельно
CACHE_INDEX_LEN	4 бита

CACHE_OFFSET_LEN	5 битов
CACHE_SIZE	2 Кбайта
CACHE_LINE_SIZE	вычислить самостоятельно
CACHE_LINE_COUNT	вычислить самостоятельно
CACHE_SETS	вычислить самостоятельно