5

Блок загрузки/сохранения (LSU)

Целью лабораторной работы является **разработка и внедрение блока загрузки и сохранения для подключения внешней памяти программ/данных**, вместо исключенных из процессора памяти программ и памяти данных.

- Функции и задачи блока загрузки/сохранения
- Интерфейс процессора и блока загрузки/сохранения (DC-LSU)
- Интерфейс блока загрузки/сохранения и памяти (LSU-MEM)

Функции и задачи блока загрузки/сохранения

Модуль загрузки и сохранения (Load/Store Unit – **LSU**, **miriscv_lsu**) служит для исполнения инструкций типа **LOAD** и **STORE**: является прослойкой между внешним устройством – памятью, и ядром процессора. **LSU** считывает содержимое из памяти данных или записывает в нее требуемые значения, преобразуя 8- и 16-битные данные в знаковые или беззнаковые 32-битные числа для регистров процессора. В процессорах с **RISC** архитектурой с помощью **LSU** осуществляется обмен данными между регистрами общего назначения и памятью данных.

Для достижения цели лабораторной работы необходимо выполнить следующий ряд задач:

– разработать блок загрузки и сохранения;

– удалить из процессора память инструкций (**Instruction Memory**), подключив вход и выход удаленной памяти (значение **PC** и **instr**) в качестве входов и выходов процессора;

– удалить из процессора память данных (**Data Memory**) и подключить вместо нее блок загрузки и сохранения (**Load/Store Unit**), выводы которого будут подключены к выводам процессора для подключения к внешней памяти;

Далее приводится прототип устройства блока загрузки и сохранения, на котором объявляются используемые входные и выходные сигналы. Часть сигналов взаимодействует с ядром процессора, часть – подключается из процессора к внешней памяти.

```
module miriscv_lsu
(
  input           clk_i,              // синхронизация
  input           arstn_i,            // сброс внутренних регистров

  // core protocol
  input   [31:0]  lsu_addr_i,         // адрес, по которому хотим обратиться
  input           lsu_we_i,           // 1 – если нужно записать в память
  input   [2:0]   lsu_size_i,         // размер обрабатываемых данных
  input   [31:0]  lsu_data_i,         // данные для записи в память
  input           lsu_req_i,          // 1 - обратиться к памяти
  output          lsu_stall_req_o,    // используется как !enable pc
  output  [31:0]  lsu_data_o          // данные считанные из памяти

  // memory protocol
  input   [31:0]  data_rdata_i,       // запрошенные данные
  output          data_req_o,         // 1 - обратиться к памяти
  output          data_we_o,          // 1 - это запрос на запись
  output  [3:0]   data_be_o,          // к каким байтам слова идет обращение
  output  [31:0]  data_addr_o,        // адрес, по которому идет обращение
  output  [31:0]  data_wdata_o,       // данные, которые требуется записать
);
```

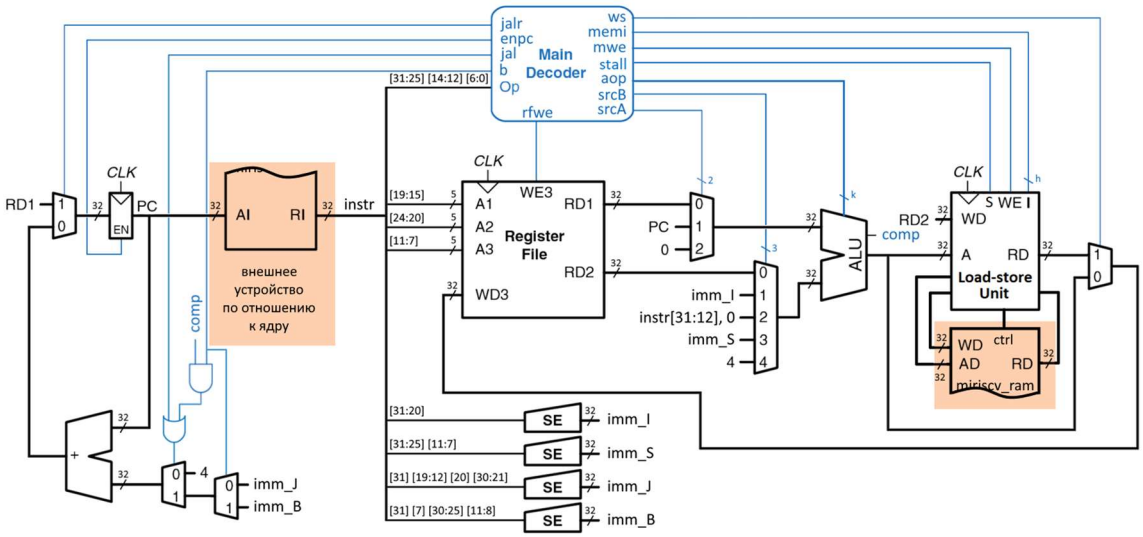
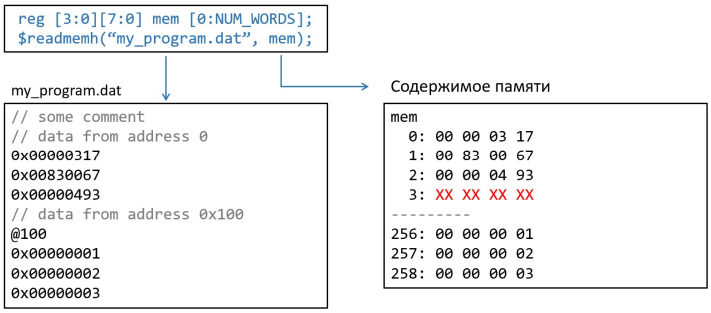


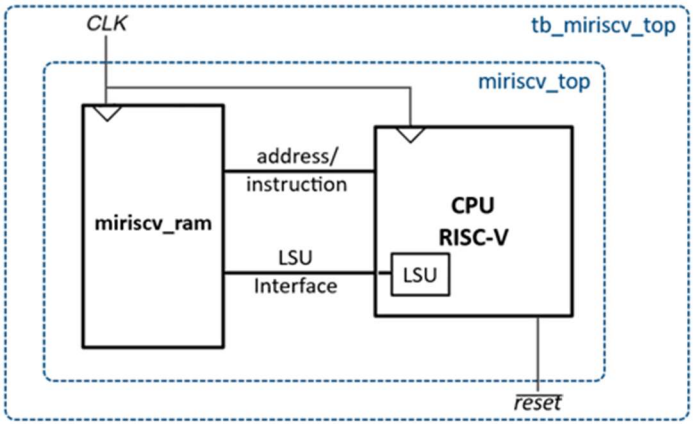
Рисунок выше демонстрирует упрощенное подключение **LSU** к общей памяти команд-данных.

Как было сказано ранее старые модули памяти **Data Memory** и **Instruction Memory** удаляются из проекта. Их роль будет играть внешняя двухпортовая память **miriscv_ram.sv** (написана на **SystemVerilog**), которая инициализируется (то есть устанавливаются ее начальные значения при подаче питания) одним файлом, содержащим и *инструкции для процессора* **RISC-V**, начинающиеся с нулевых адресов, и *статические данные*, расположенные далее. Используя специальный синтаксис можно указывать в фале адреса областей памяти, которые требуется инициализировать.



Процессор соединяется с памятью через **LSU**, однако, из-за того, что один из портов памяти (для считывания инструкций) используется только для операций чтения, то его входы и выходы присоединяются к **miriscv_ram** напрямую, а не через **LSU**.

К дополнительным материалам лабораторной работы так же относится файл верхнего уровня иерархии **miriscv_top.sv**, который описывает (и демонстрирует) подключение процессора к памяти. После реализации и добавления блока LSU в процессор его входы и выходы должны совпадать со входами и выходами, описанными в модуле **miriscv_top**.



Внимательно изучи**!** как все организовано в модуле **miriscv_top**, и как в нем подключаются модули процессора и памяти. На рисунке упрощенное подключение модулей внутри **miriscv_top**.

Для того, чтобы верифицировать работу системы память-процессор, дополнительные материалы к лабораторной включают в себя файл тестового окружения **tb_miriscv_top.v**. Модуль содержит два параметра: **RAM_SIZE** (указывает размер **miriscv_ram** в байтах) и **RAM_INIT_FILE** (указывает имя txt-файла для инициализации памяти).

Для проверки необходимо написать осмысленную программу, считывающую *из*, и записывающую *в* память байты и полуслова с расширением знака и без. Например, обработка массива.

Интерфейс процессора и блока загрузки/сохранения

Параграф посвящен описанию сигналов и правил взаимодействия между процессором, в частности его основным дешифраторов **DC**, и блоком загрузки/сохранения **LSU** (**core protocol**).

На входной порт **lsu_addr_i** от процессора поступает адрес ячейки памяти, к которой будет произведено обращение. Намеренье процессора обратиться к памяти (и для чтения, и для записи) отражается выставлением сигнала **lsu_req_i** в единицу. Если процессор собирается записывать в память, то сигнал **lsu_we_i** выставляется в единицу, а сами данные, которые следует записать, поступают от него на вход **lsu_data_i**. Если процессор читает из памяти, то сигнал **lsu_we_i** находится в нуле, а считанные данные подаются для процессора на выход **lsu_data_o**.

Инструкции **LOAD** и **STORE** в **RV32I** поддерживают обмен 8-битными, 16-битными или 32-битными значениями, однако в самом процессоре происходит работа только с 32-битными числами, поэтому загружая байты или полуслова из памяти их необходимо предварительно расширить до 32-битного значения. Для выбора разрядности на вход **LSU** подается сигнал **lsu_size_i**, принимающий следующие значения:

Название	Значение	Пояснение
LDST_B	3'd0	Знаковое 8-битное значение
LDST_H	3'd1	Знаковое 16-битное значение
LDST_W	3'd2	32-битное значение
LDST_BU	3'd4	Беззнаковое 8-битное значение
LDST_HU	3'd5	Беззнаковое 16-битное значение

Формат представления числа (является оно знаковым или беззнаковым) имеет значение только для операций типа **LOAD**: если число знаковое, то производится расширение знака (**sign extension** – операция добавления знакового бита числа со стороны его старшего разряда) до 32 бит, а если беззнаковое – расширение нулями (**zero extension** – операция добавления нулевого бита со стороны старшего разряда числа с целью увеличения разрядности).

Операции расширения нулями или знаком позволяют увеличить разрядность числа без изменения его значения. Разберем следующий пример: требуется увеличить разрядность 8-битного слова **S** до 32 бит так, чтобы полученное слово **L** сохранило первоначальное значение. Если **S** – знаковое, производим расширение знака следующим образом:

```
wire signed [7:0]  S;
wire signed [31:0] L;

assign L = { { 24{S[7]} }, S};
```

Если S – беззнаковое, производим расширение нуля следующим образом:

```
wire [7:0]  S;
wire [31:0] L;

assign L = {24'b0, S};
```

Для операций типа **STORE** формат представления чисел не важен, для них **lsu_size_i** сможет принимать значение только от 0 до 2.

Выходной сигнал **lsu_stall_req_o** нужен для управления входом **enable pc**. Мы точно знаем, что если на текущем такте был выставлен **data_req_o**, то на следующем такте данные будут записаны/считаны из памяти. Соответственно **data_req_o**, как и **lsu_stall_req_o** достаточно поднять на один такт: на следующем такте считанные данные уже будут у нас и мы будем готовы поместить их в регистровый файл.

Интерфейс блока загрузки/сохранения и памяти

В параграфе описывается организация внешней памяти, и то, как к ней подключаются **LSU**, **pc** и **instr**.

Память данных имеет 32-битную разрядность ячейки памяти (слово, word) и поддерживает побайтовую адресацию. Это значит, что существует возможность записи значения по одному байту в пределах одного слова (4-байтовой ячейки памяти). Для указания на необходимые байты интерфейс к памяти предусматривает использование 4-битного сигнала **data_be_o**, подаваемого вместе с адресом слова **dara_addr_o**. Позиции битов 4-битного сигнала соответствуют позициям байтов в слове. Если конкретный бит **data_be_o** равен 1, то соответствующий ему байт будет записан в память. Данные для записи подаются на выход **data_wdata_o**. На результат чтения из памяти состояние **data_be_o** не влияет, так как чтение производится всегда по 32-бита.

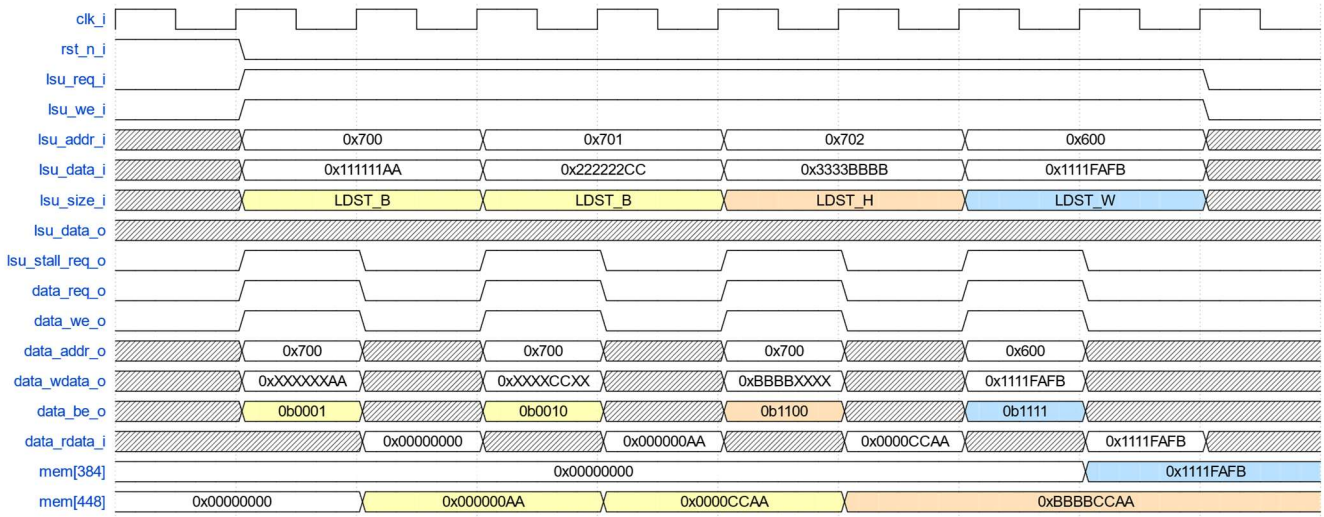
После получения запроса на чтение/запись из ядра, **LSU** перенаправляет запрос в память данных, реагируя на приходящие от нее сигналы. Сигнал **data_req_o** сообщает памяти о наличии запроса. Сигнал **data_we_o** сообщает памяти о типе этого запроса: **data_we_o** равен 1, если отправлен запрос на запись, 0 – если отправлен запрос на чтение.

LSU должен поднять сигнал **lsu_stall_req_o** – это сообщит блоку управления, что работа ядра должна быть приостановлена, пока запрос не будет выполнен (об этом уже писалось ранее). Сигнал **data_rdata_i** содержит данные из ячейки памяти на момент принятия запроса. Следовательно, после совершения записи в память **data_rdata_i** будет хранить *предыдущее* значение из ячейки, а не записанное.

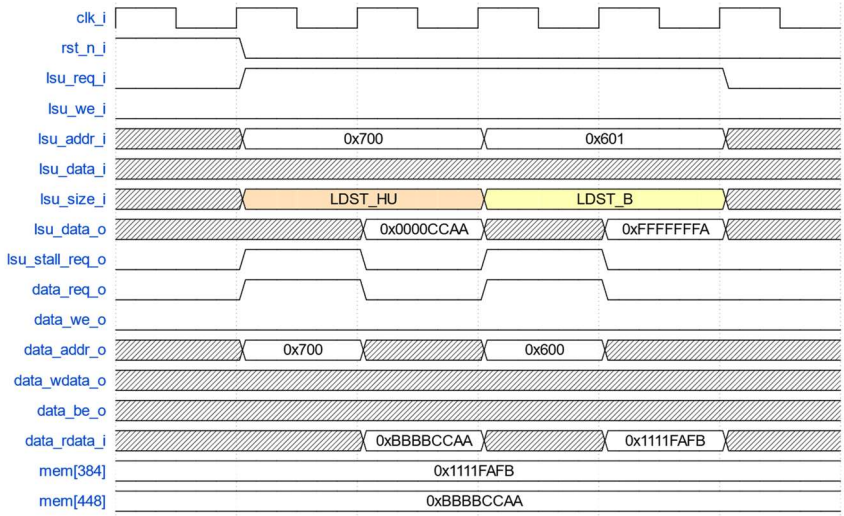
Команды	Byte Offset	lsu_data_o
lb	00	`{{24{data_rdata_i[7]}, data_rdata_i[7:0]}`
	01	`{{24{data_rdata_i[15]}, data_rdata_i[15:8]}`
	10	`{{24{data_rdata_i[23]}, data_rdata_i[23:16]}`
	11	`{{24{data_rdata_i[31]}, data_rdata_i[31:24]}`
lh	00	`{{16{data_rdata_i[15]}, data_rdata_i[15:0]}`
	10	`{{16{data_rdata_i[31]}, data_rdata_i[31:16]}`
lw	00	`data_rdata_i[31:0]`
lbu	00	`{24'b0, data_rdata_i[7:0]}`
	01	`{24'b0, data_rdata_i[15:8]}`
	10	`{24'b0, data_rdata_i[23:16]}`
	11	`{24'b0, data_rdata_i[31:24]}`
lhu	00	`{16'b0, data_rdata_i[15:0]}`
	10	`{16'b0, data_rdata_i[31:16]}`

Команды	Byte Offset	data_wdata_o	data_be_o
sb	00	`{ 4{lsu_data_i[7:0]} }`	0001
	01		0010
	10		0100
	11		1000
sh	00	`{ 2{lsu_data_i[15:0]} }`	0011
	10		1100
sw	00	`lsu_data_i[31:0]`	1111

Запись в память



Чтение из памяти



Память для инструкций

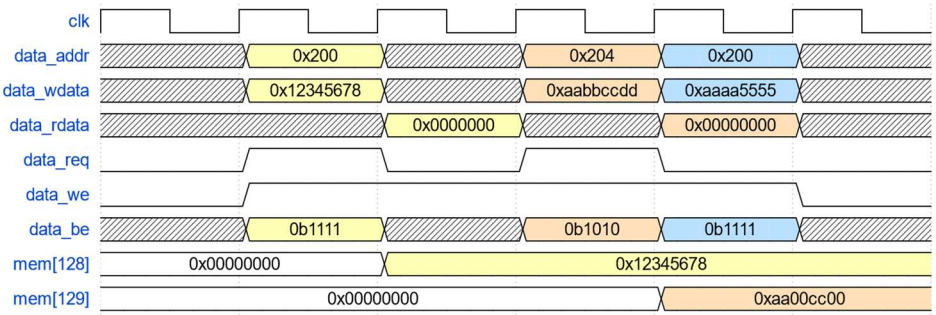
1. Разрядность инструкций – 32 бита.
2. Память инструкций доступна только для чтения.
3. Для чтения необходимо выставить требуемый адрес на шине **instr_addr_o**, после чего на этом же такте данные из памяти появятся на шине **instr_rdata_i**.

Память для данных

1. Разрядность данных – 32 бита.
2. Память данных доступна для чтения и записи.
3. Для обращения к памяти как для чтения, так и для записи сигнал **data_req_o** должен быть установлен в логическую 1.
4. В случае записи сигнал **data_we_o** должен быть установлен в логическую 1, в случае чтения – в логический 0.
5. Для записи необходимо выставить требуемый адрес на шине **data_addr_o**, выставить записываемые данные на шине **data_wdata_o** и установить **data_req_o** и **data_we_o** в 1. Данные будут записаны на следующем такте.
6. Для чтения необходимо выставить требуемый адрес на шине **data_addr_o** и установить **data_req_o** в 1, после чего на следующем такте данные из памяти появятся на шине **data_rdata_i**. При этом считанные данные будут соответствовать содержимому памяти на предыдущем такте. Так, на следующем такте после запроса на запись в память будет записано значение из **data_wdata_o**, а на шине **data_rdata_i** будет установлено старое значение до записи.
7. Поддерживается побайтовая запись в память с помощью сигнала data_be_o. Если при запросе на запись биты сигнала **data_be_o** установлены в 1, то соответствующие им байты из **data_wdata_o** будут записаны в память. Если в 0 – соответствующие байты игнорируются при записи. Младший бит data_be_o соответствует младшему байту **data_wdata_o**, старший бит – старшему.

Далее приведены временные диаграммы работы с памятью.

Транзакция записи



Транзакция чтения