Архитектура GPU

В рамках студенческой стажировки стажёрам предлагается реализовать графический (массивно-параллельный) процессор — Graphics Processing Unit (GPU).

Архитектура системы представлена на рисунке 1.

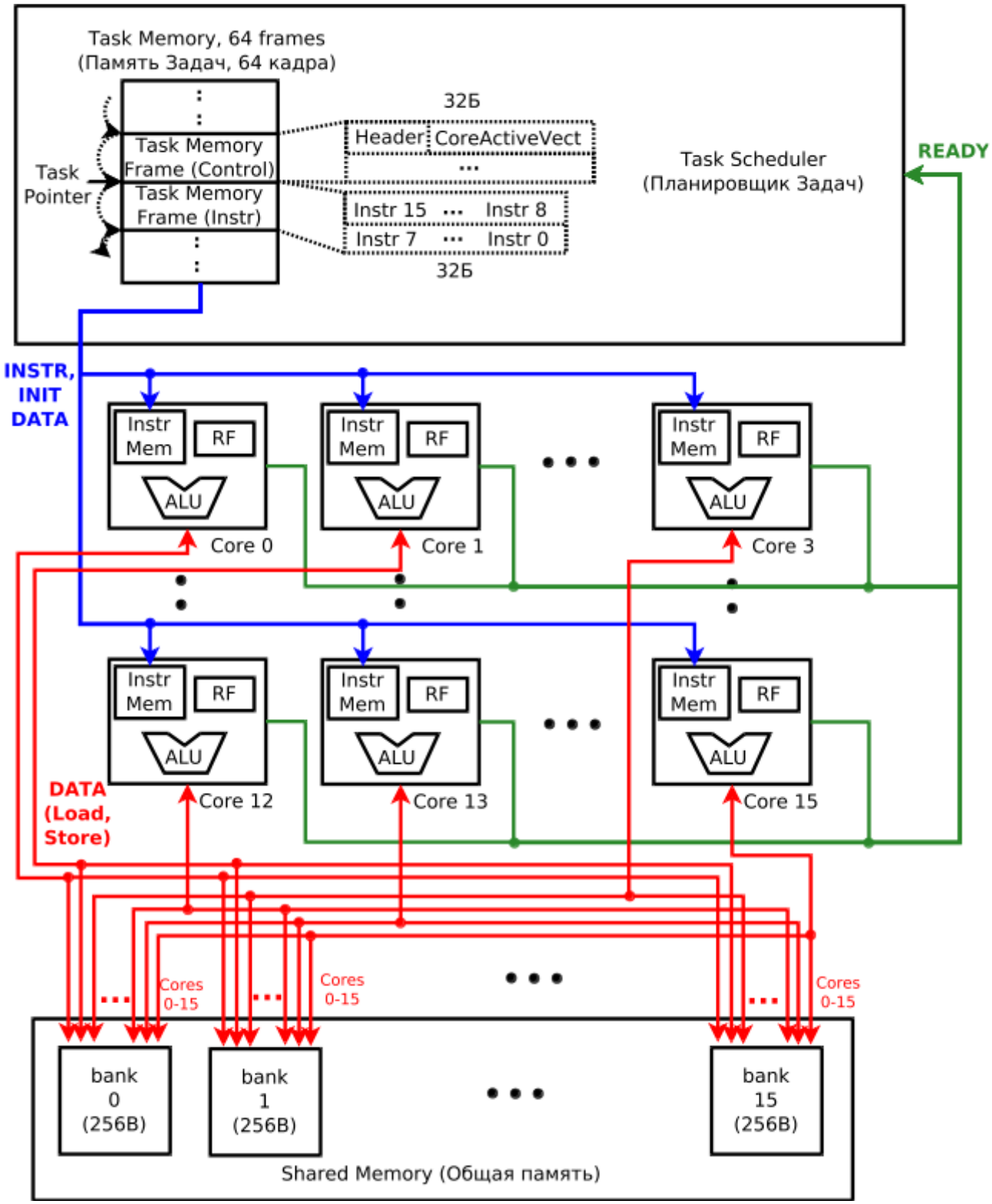

Рисунок 1: Архитектура GPU

Предлагаемый для реализации GPU напоминает по архитектуре типовой элемент полноценных повсеместно используемых GPU, но при этом значительно упрощён. Основные устройства в составе GPU:

- Планировщик Задач (Task Scheduler, TS) — устройство, управляющее работой GPU. Содержит Память Задач (Task Memory — TM), содержащую код *программы* (*шейдера*) GPU, разбитый на отдельные *задачи*, а также управляющую информацию и инициализиационные данные. Планировщик задач последовательно проходит по содержимому TM, выдавая задачи на исполнение ядрам системы, организованным в группы (для каждой задачи либо для нескольких задач). Устройство TS также отвечает за синхронизацию выданных на исполнение задач.
- Ядра (Cores) — основные вычислительные элементы, 16 штук. Каждое активное ядро исполняет код задачи, присылаемый TS в начале каждого цикла исполнения задачи. Ядро представляет собой RISC-подобную машину с системой команд (СК), описанной ниже в п. «Система команд ядра». СК ядра содержит типовые операции над регистровым файлом и памятью: арифметику, чтение / запись из памяти, переходы, а также операцию ready, сигнализирующую блоку TS об окончании исполнения задачи ядром посредством выдачи одноимённого сигнала READY.
- Общая память (Shared Memory, SM) — память в составе GPU, общая для всех ядер. Ядра не имеют собственной памяти - все операции чтения и записи, выполняемые ядрами, обрабатываются в SM, которая расслоена на 16 банков. У каждого ядра есть доступ к любому банку SM. При старте работы GPU память SM инициализируется извне некоторыми исходными данными задачи.

Форматы кадров задач

Блоки данных в памяти TM блока TS, соответствующие задачам, называются кадрами памяти задачи (TMF - Task Memory Frames). Они имеют фиксированный размер 32Б и бывают двух типов:

- Кадр команд (IF, Instruction Frame) — кадр кода задачи - блок из 16 инструкций, рассылаемый всем активным для данной задачи ядрам для исполнения.
- Управляющий кадр (CF, Control Frame) — блок управляющей информации; содержит, в числе прочего, следующие управляющие поля:
  - IF_Num[5:0] - число предстоящих кадров команд;

Примечание: Поле IF_Num требуется для того, чтобы при декодировании отличать кадры разных типов (CF от IF), а также не отводить в кадре IF служебных полей (все поля кадра IF можно использовать под инструкции); После кадра управления должны следовать IF_Num кадров команд; если это правило нарушено, поведение не определено.

  - CoreActiveVect[15:0] — маска активных ядер задачи;
  - fence[1:0] — барьерность для задач предстоящих кадров команд:
    - 2'h0 — no, нет барьерной семантики;
    - 2'h1 — acq, барьер acquire (следующие задачи не ставятся на исполнение, пока не завершится эта задача);
    - 2'h2 — rel, барьер release (эта задача не начинается, пока есть прошлые незавершённые задачи);
    - 2'h3 — резерв;

Цикл работы задач

Каждая задача (фрагмент шейдера GPU), упакованная в кадр памяти TM планировщика задач TS, ставится на исполнение на группе ядер (размером от одного до всех доступных ядер), которая задаётся специальной маской активных ядер. Ядра выбранной группы стартуют

синхронно, исполняют один и тот же код задачи и работают параллельно. Код задачи ограничен 16-ю инструкциями ядра.
Исполнение кадра IF одной задачи называется *циклом работы задачи*, который состоит из следующих стадий:

- Пересылка кода задачи из устройства TS ядрам, указанным в маске активных ядер (группе ядер);
- Синхронный старт работы всех ядер группы по окончании рассылки кода;
- Исполнение кода на ядрах группы;
- Каждое ядро группы завершает исполнение (в общем случае — в разное время), отправляя при этом сигнал READY в устройство TS;

Задача считается завершённой, если исполнение её кода завершилось на всех ядрах выбранной группы, и TS собрал сигналы READY от всех ядер группы.

Исполнение кадра CF сводится к модификации состояния устройства TS и не влечёт за собой исполнение кода на ядрах, в отличие от исполнения кадров IF, которые, наоборот отвечают за исполнение кода на ядрах.

Синхронизация работы задач

С точки зрения программиста кадры должны выдаваться на исполнение **в программном порядке**. При этом синхронизация последовательно расположенных в памяти TM задач может быть организована различным образом:

- При отсутствии барьерной сематники, если текущая задача поставлена на исполнение и ещё не завершилась, но есть свободные ядра, а также есть следующая задача, маска активных ядер которой не пересекается с прошлой задачей (или, в более общем случае — с масками прошлых выданных на исполнение, но не завершённых задач), то следующая задача также может быть выдана на исполнение. Проще говоря, если ни одно из ядер, запрашиваемых следующей задачей, не занято, эту задачу можно выдать на исполнение.
- Если же кадры текущей или следующей задач имеют барьерную семантику, или же, при отсутствии барьерной семантики, кадры задач пересекаются между собой по маске активных ядер, появляются ограничения на параллельное исполнение кадров (сериализация).

Исполнение очередного кадра CF может быть выполнено не дожидаясь окончания исполнения прошлого кадра при условии, что сохраняется семантика модификации состояния TS в программном порядке

Рассмотрим примеры различных типов синхронизации кадров.
Пример 1. В TM содержится такая последовательность кадров:

- CF0(IF_Num = 4, CoreActiveVect=16'h0f0f, fence=**acq**),
- {IF0, IF1, IF2, IF3},
- CF1(IF_Num = 2, CoreActiveVect=16'hf0f0, fence=**acq**)
- {IF4, IF5}

то исполнение будет поделено на 8 циклов:

Цикл 1: Исполнение CF0 с активацией ядер по маске (активны ядра с номерами 0-3 и 8-11); ожидается IF_Num=4 кадров команд;

Циклы 2-5: Исполнение IF_Num=4 кадров команд IF0-IF3 на активированных в первом цикле 1 ядрах; при этом циклы 2-5 идут строго последовательно (т. к. имеют одинаковую маску активных ядер), а цикл 6 также не начинается, пока не закончится цикл 5 (исполнение IF3) из-за барьерной семантики acq кадра CF0, распространяющейся на кадры

IF0-IF3.

Цикл 6: Исполнение CF1 с активацией ядер по маске (активны ядра с номерами 4-7 и 8-15); ожидается IF_Num=2 кадра команд.

Циклы 7-8: Исполнение IF_Num=2 кадров команд IF4-IF5 на активированных в цикле 6 ядрах; при этом циклы 7-8 идут строго последовательно (т. к. имеют одинаковую маску активных ядер).

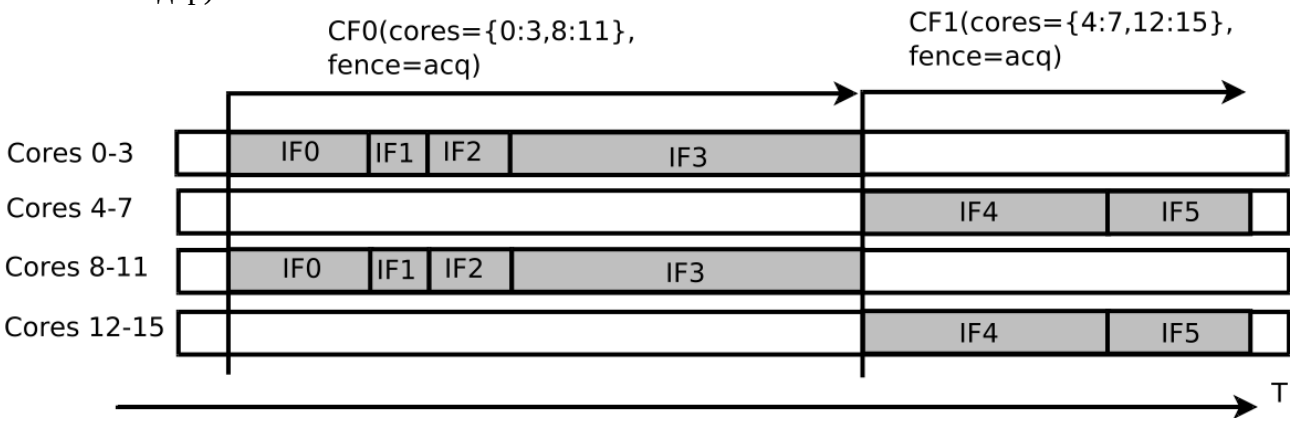

Рисунок 2. Временная диаграмма исполнения примера 1

Пример 2. Этот пример отличается от примера 1 барьерностью кадра CF0:

- CF0(IF_Num = 4, CoreActiveVect=16'h0f0f, fence=**no**),
- {IF0, IF1, IF2, IF3},
- CF1(IF_Num = 2, CoreActiveVect=16'hf0f0, fence=**acq**)
- {IF4, IF5}

то исполнение будет поделено на 8 циклов, часть из которых накладываются друг на друга:

Цикл 1: Исполнение CF0 с активацией ядер по маске (активны ядра с номерами 0-3 и 8-11); ожидается IF_Num=4 кадров команд;

Циклы 2-5: Исполнение IF_Num=4 кадров команд IF0-IF3 на активированных в первом цикле 1 ядрах; при этом циклы 2-5 идут строго последовательно (т. к. имеют одинаковую маску активных ядер).

Цикл 6: Исполнение CF1 с активацией ядер по маске (активны ядра с номерами 4-7 и 8-15); ожидается IF_Num=2 кадра команд. При этом исполнение цикла 6 может быть выполнено параллельно (не дожидаясь окончания) с исполнением цикла 5, ведь барьерная семантика кадров IF0-IF3 в CF0 отсутствует, а пересечения по маске активных ядер нет.

Циклы 7-8: Исполнение IF_Num=2 кадров команд IF4-IF5 на активированных в цикле 6 ядрах; при этом циклы 7-8 идут строго последовательно, но циклы 7-8 исполняются параллельно с циклами 2-5.

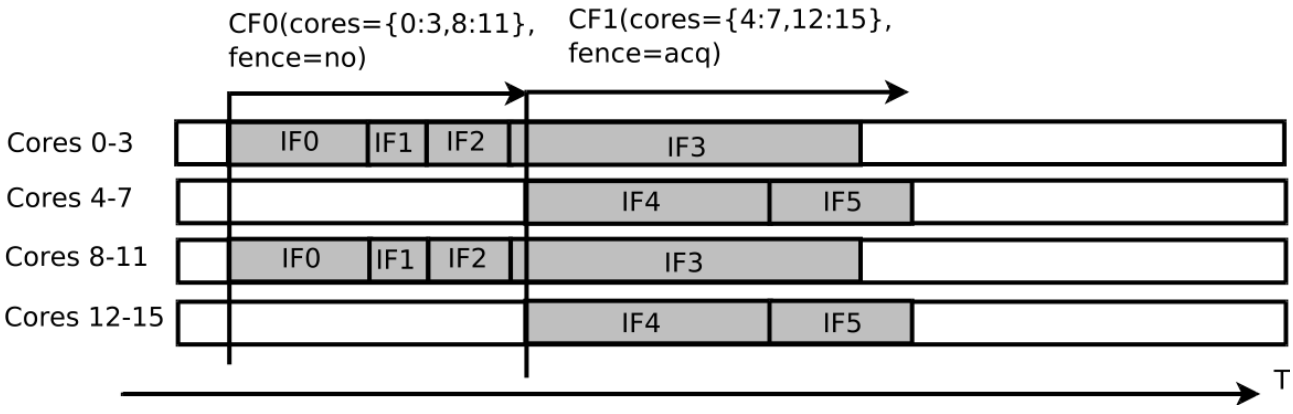

Рисунок 3. Временная диаграмма исполнения примера 2

Система команд планировшика (Task Scheduler).

TM — Task Memory, память задач на 1024 команды (???Б — управление).

TMF — Task Memory Frame, кадр памяти задач — блок данных, содежащий в себе всю информацию, необходимую для очередного запуска задачи; размер кадра фиксированный — 32Б.

Форматы кадра задачи в памяти TM

- Format 1: Control Frame (кадр управления)

	Line bytes [7-6]	**Line bytes [5-4]**	**Line bytes [3-2]**	**Line bytes [1-0]**
Task line 3 (bytes [31-24])	{Init_R0[15][7:0], Init_R0[14][7:0]}	{Init_R0[13][7:0], Init_R0[12][7:0]}	{Init_R0[11][7:0], Init_R0[10][7:0]}	{Init_R0[9][7:0], Init_R0[8][7:0]}
Task line 2 (bytes [23-9])	{Init_R0[7][7:0], Init_R0[6][7:0]}	{Init_R0[5][7:0], Init_R0[4][7:0]}	{Init_R0[3][7:0], Init_R0[2][7:0]}	{Init_R0[1][7:0], Init_R0[0][7:0]}
Task line 1 (bytes [15-8])	*Резерв*	*Резерв*	*Резерв*	*Резерв*
Task line 0 (bytes [7-0])	*Резерв*	Init_R0_Vect[15:0]	Core_Active_Vect[15:0]	Header[15:0]

  - Init_R0[15:0][7:0] — значения, которыми инициализируются регистры R0 активных ядер (от 0-го до 15-го) при исполнении кадра CF;
  - Init_R0_Vect[15:0] — вектор инициализации регистров R0 активных ядер значениями Init_R0[15:0][7:0]; если i-й разряд вектора установлен, для i-го ядра выполняется инициализация регистра R0;
  - Header[15:0] — заголовок кадра
    - Header[15:8] - резерв;
    - Header[7:6] = fence[1:0], барьерность для задач предстоящих кадров команд:
      - 2'h0 — no, нет барьерной семантики;
      - 2'h1 — acq, барьер acquire (следующие задачи не ставятся на исполнение, пока не завершится эта задача);
      - 2'h2 — rel, барьер release (эта задача не начинается, пока есть прошлые незавершённые задачи);
      - 2'h3 — резерв;
    - Header[5:0] = IF_Num[5:0], число предстоящих кадров команд;
  - Core_Active_Vect[15:0] - вектор активных ядер; каждый разряд соответствует номеру активного ядра («1» - активно, «0» - не активно); если ядро активно, код данной задачи загружается и исполняется на этом ядре; если ядро неактивно, в данном раунде вычислений (запуске задачи) это ядро не участвует;

- Format 2: Instruction Frame (кадр команд)

	Line bytes [7-6]	**Line bytes [5-4]**	**Line bytes [3-2]**	**Line bytes [1-0]**

Task line 3 (bytes [31-24])	Instruction_15 [15:0]	Instruction_14 [15:0]	Instruction_13 [15:0]	Instruction_12 [15:0]
Task line 2 (bytes [23-9])	Instruction_11 [15:0]	Instruction_10 [15:0]	Instruction_9 [15:0]	Instruction_8 [15:0]
Task line 1 (bytes [15-8])	Instruction_7 [15:0]	Instruction_6 [15:0]	Instruction_5 [15:0]	Instruction_4 [15:0]
Task line 0 (bytes [7-0])	Instruction_3 [15:0]	Instruction_2 [15:0]	Instruction_1 [15:0]	Instruction_0 [15:0]

    - Instruction_15-0[15:0] — команды программы ядра в данной задаче; все команды загружаются в память команд каждого ядра, заданного в векторе активных ядер Core_Active_Vect[15:0] последнего исполненного управляющего кадра

Регистры.

- TP[5:0] — Task Pointer, указатель задачи;
- EXEC_MASK[15:0] — статусный регистр-маска, в который при старте задачи записывается маска активных ядер; по завершении каждой зачади ядро исполняет команду ready, которая устанавливает соответствующий интерфейсный сигнал, выдаваемый ядром в Task Scheduler; при получении сигнала разряд EXEC_MASK[15:0], соответствующий готовому ядру, сбрасывается в «0». При EXEC_MASK[15:0] == 16'h0 исполнение задачи (текущего кадра команд) завершено.

Система команд ядра (Core).

Регистры.

- IP[3:0] — Instruction Pointer, указатель команды;
- RF[15:0][7:0] — регистровый файл, 16 регистров по 1Б каждый:

R0[7:0]
R1[7:0]
:
:
R14[7:0]
R15[7:0]

Форматы команды.

Размер команд (instr_size) фиксированный — 2Б;

Ins	[15:12]	[11:8]	[7:4]	[3:0]
F1	opc[3:0]	src_0[3:0]	src_1[3:0]	dst[3:0]
F2	opc[3:0]	const[7:4]	const[3:0]	dst[3:0]
F3	opc[3:0]	src_0[3:0]	src_1[3:0]	src_2[3:0]
F4	opc[3:0]	src_0[3:0]	target[3:0]	-

- opc[3:0] — код операции (см. далее);
- src_0,1,2[3:0] — номер входного операнда (один из регистров R0-R15)
- dst[3:0] — номер выходного операнда (один из регистров R0-R15)
- const[7:0] — операнд-литерал (константа), закодированный в команде;
- target[3:0] — IP целевой команды, используется в команде bnz;

Команды.

- F1:
  - (opc=0x0) **nop**
    - No operation
  - (opc=0x1) **add** src0, src1, dst
    - RF[dst] <- RF[src_0] + RF[src_1]
  - (opc=0x2) **sub** src0, src1, dst
    - RF[dst] <-RF[src_0] — RF[src_1]
  - (opc=0x3) **mul** src0, src1, dst
    - {RF[dst+1], RF[dst]} <-RF[src_0] * RF[src_1]
      (если dst=15, dst+1=0)
  - (opc=0x4) **div** src0, src1, dst
    - RF[dst] <-RF[src_0] / RF[src_1]
  - (opc=0x5) **cmpge** src0, src1, dst
    - RF[dst] <- RF[src_0] >= RF[src_1]
  - (opc=0x6) **rshft** src0, src1, dst
    - RF[dst] <- RF[src_0] >> src_1[2:0]
  - (opc=0x7) **lshft** src0, src1, dst
    - RF[dst] <- RF[src_0] << src_1[2:0]
  - (opc=0x8) **and** src0, src1, dst
    - RF[dst] <- RF[src_0] & RF[src_1]

- (opc=0x9) **or** src0, src1, dst
  - RF[dst] <- RF[src_0] | RF[src_1]
- (opc=0xa) **xor** src0, src1, dst
  - RF[dst] <- RF[src_0] ^ RF[src_1]
- (opc=0xb, src_1[7:6]==2'h0) **ld** [src0, src1], dst
  - RF[dst] <- MEM[ADDR][11:0]],
    где ADDR[11:0] = {RF[src_1][3:0]; RF[src_0][7:0]}
- (opc=0xb, src_1[7:6]==2'h1) **ld_sync** [src0, src1], dst
  - аналогично операции **ld**, но дополнительно выполняется аппаратная блокировка фрагмента памяти размером 1Б (см. п. «Атомарная последовательность»);

- F2:
  - (opc=0xc) **set_const** const[7:0], dst
    - if(dst[3] = 0)
      RF[dst] <- {4'h0, core_id[3:0]}, где core_id — номер текущего ядра;
      Примечание: в этом режиме для записи доступны только регистры RF[0-7]
      else
      RF[dst] <- {const_[7:4], const[3:0]}
      Примечание: в этом режиме для записи доступны только регистры RF[8-15]
- F3:
  - (opc=0xd, src_1[7:6]==2'h0) **st** [src0, src1], src2
    - MEM[ADDR[11:0]] <- RF[src_2],
      где ADDR[11:0]={RF[src_1][3:0]; RF[src_0][7:0]}
  - (opc=0xd, src_1[7:6]==2'h1) **st_sync** [src0, src1], src2
    - аналогично операции **st**, но дополнительно выполняется аппаратная разблокировка фрагмента памяти размером 1Б (см. п. «Атомарная последовательность»);
- F4:
  - (opc=0xe) **bnz** disp, src0
    - if(src0 != 0)
      IP ← target[3:0] * instr_size
      else
      IP ← IP + instr_size
  - (opc=0xf) **ready**
    - IP<-0; конец работы ядра (выдача сигнала READY в Task Scheduler)

Существует два сценария завершения исполнения кода задачи на данном ядре:

1. Исполнение команды ready
2. Исполнение команды с IP=instr_size*0xf, не являющейся переходом (окончание кода задачи в памяти команд).

Во всех перечисленных выше ситуациях ядро завершает исполнение задачи, отправляя устройству TS сигнал READY.

Общая память (Shared Memory).

Общая память, или SM (рис. 1) организована в виде 16 банков глубиной 256 строк, каждая строка шириной 1 байт. Каждое ядро может обращаться в SM по выделенному интерфейсу, и каждому ядру доступен любой из 16 банков.

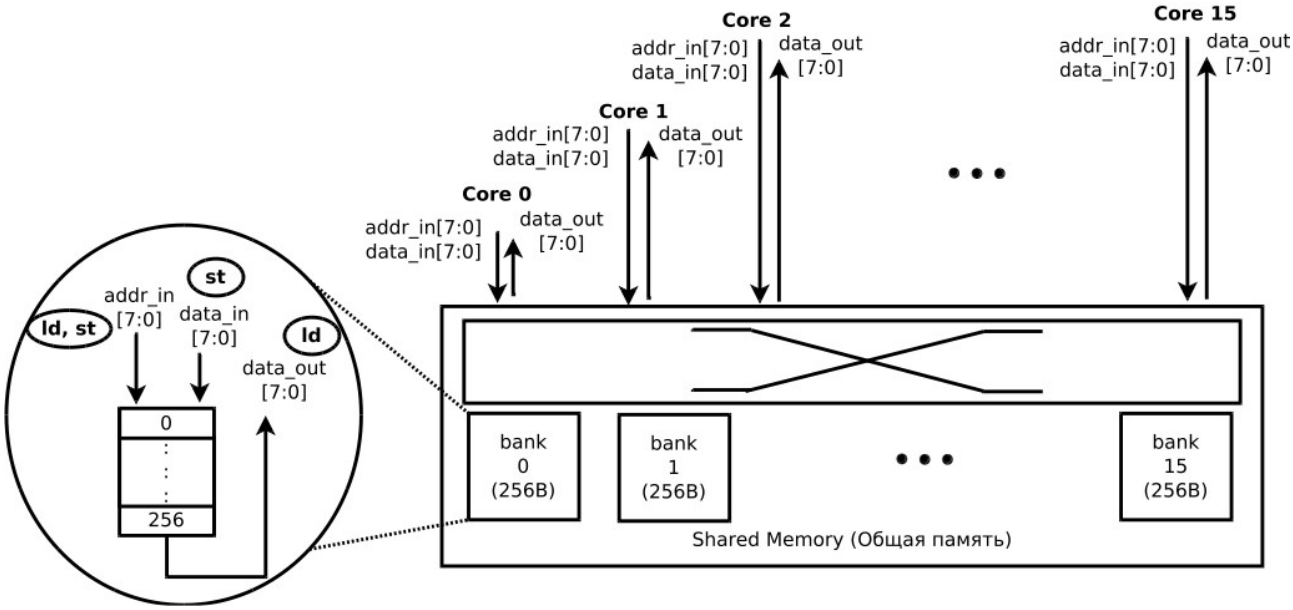

Рисунок 4. Общая память.

Атомарная последовательность.

Для реализации синхронизации работы задач на разных ядрах (поток управления задачи, запущенной на каждом конкретном ядре, будем называть просто *потоком*) через общую память предусмотрен механизм атомарной последовательности операций обращения в память. Атомарная последовательность в наиболее общем виде:

:

ld_sync [ADDR0] → dst

<

:

тело атомарной последовательности

:

>

st_sync [ADDR0], data

:

Свойства атомарной последовательности:

1. Последовательность всегда открывается операцией ld_sync, а закрывается операцией st_sync.
2. Весь код последовательности должен заключаться внутри одного кадра команд.
3. Запрещены вложенные атомарные последовательности.
4. Тело последовательности не должно содержать обращений в память, операций ready и выхода за пределы IP=0xf (ситуации с выдачей сигнала READY по окончании значимых команд для данного ядра).
5. Начиная с выполнения операции ld_sync и вплоть до выполнения операции st_sync (не включая последнее) фрагмент общей памяти (байт), к которому идёт обращение, аппаратно блокируется для всех прочих потоков. Если потоков, обращающихся атомарно или неатомарно к данному фрагменту, несколько (конкурентный доступ), то аппаратно выбирается какой-то один поток, который мы назовём *победившим* потоком, и он блокирует доступ остальных потоков до выполнения операции st_sync, снимающей блокировку.
7. В случае конкурентного доступа нескольких потоков к одному фрагменту, операции обращения в память (как ld*, так и st*) потоков, не являющихся победившими, аппаратно

блокируются, т. е. не завершаются до снятия блокировки победившим потоком.
8. Поддерживается единственная атомарная последовательности на каждый банк общей памяти; таким образом, возможно одновременное выполнение до 16 атомарных последовательностей при условии, что каждая последовательность выполняется по своему банку.