**ШИНЫ**

**Адресная шина (АВ)**

|  |  |
| --- | --- |
| 2 | 12 |
| Номер устройства | Адрес |

Первые два бита отвечают за выбор устройства

0 – ничего

1 – RAM

2 – RCM

3 – DMA settings

**Контрольная шина (СВ)**

CB[7..0]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | 0 – ничего  1 - выставить регистр данных на шину | 0 – ничего  1 – записать в AR с шины данных | 0 – ничего  1 – выставить AR на шину адреса | 0 – чтение  1 - запись |

**Конвейерная шина**

CB\_convey[7..0]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | HLT в конце цикла | Предсказание было верное | Конфликт чтения записи | WRITE ready | EXEC ready | OF ready | IF ready |

CB\_convey[15..8]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Reg[63] из IF | 0 – ничего  1 – остановить WRITE | 0 – ничего  1 – остановить OF and EXEC | 0 – ничего  1 – перезаписать адрес IF на правильный | Сменить регистры конвейера | Перезаписать регистры конвейера | Запустить конвейер WRITE | Запустить конвейер IF & OF |

**Внутренняя управляющая шина (ICB) и внутренние шины блоков.**

Контролирует направление связи всех внутренних шин с основными. Внутренняя память блока, содержащего микрокоманды имеет ширину 48 бит: ICB[15..0] CB[15..0] AB[15..0].

Младший байт ICB управляет связями всех внутренних шин. ICB[7..0]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Память микрокоманд в шины блока | | Шины блока в общие шины | | | Общие шины в шины блока | | |
| CB | AB | CB | AB | DB | CB | AB | DB |
| 0 – линия разорвана  1 – линия связана в указанном направлении | | | | | | | |

Старший байт ICB отвечает за различные взаимодействия внутри конкретного блока. В частности старший бит ICB[15] обычно используется для защелкивания входных данных (статическое управление).

**RAM**

Чтение происходит за два такта. Первый такт сменяется адрес, второй так – на шину выставляются правильно прочитанные данные.

**СТЕК**

Стек включен внутрь EXEC блока и имеет отдельную шину управления CB\_stack[15..0] и шину данных DB\_stack[15..0]. Может использоваться исключительно в двух командах (PUSH POP).

Значения бит CB\_stack[15..0]:

|  |  |
| --- | --- |
| Младшие биты | Микрокоманда |
| 0 | - |
| 1 | Увеличить указатель стека |
| 2 | Уменьшить указатель стека |
| 3 | Записать данные в стек |
| 4 | Прочитать данные из стека |
| 5 | Прочитать регистр флагов из стека |

Т.к. стек реализован на основе RAM, то он имеет все недостатки причисляемые к этому блоку (чтение следующего значения за 2 такта = смена адреса + чтение)

Указатель стека всегда имеет номер первой свободной ячейки.

**Извлечение инструкций, Instruction Fetch (IF)**

В reg[39..0] ложится полная инструкция. В reg[40] сохраняется текущее предсказание.

Если команда являлась прыжком, то второй операнд перезаписывается блоком на адрес, который не был выбран для возможности последующей поправки.

Блок имеет следующую последовательность операций:

|  |  |  |
| --- | --- | --- |
| Номер ICB | Фронт Срез | Значение |
| 8 | / | Сохранить прочитанную инструкцию (первая половина такта) |
| 9 | \ | В зависимости от предсказания сменить текущий адрес |

Необходимо учесть, что память с инструкциями (IROM) имеет динамическое управление. Переключение инструкции происходит по фронту каждого такта.

Также блок имеет возможность заменить текущий адрес по внешнему сигналу.

**Контрольный модуль (СМ)**

Включает в себя бесконечный итератор, который проходит следующие шаги:

|  |  |
| --- | --- |
| Номер шага | Значение |
| 0 | Запустить все стадии конвейера |
| 1 | Сохранить результаты каждой стадии |
| 2 | Сменить промежуточные регистры |

**Промежуточные регистры**

reg[63]: 1 – работаем 0 – пауза

IF -> OF AND EXEC

|  |  |  |
| --- | --- | --- |
| 63..41 | 40 | 39..0 |
| - | предсказание | Прочитанная инструкция |

OF AND EXEC -> WRITE

|  |  |  |  |
| --- | --- | --- | --- |
| 59..56 | 55..48 | 47..32 | 31..0 |
| Старшие 4 бита команды | Флаги операции | Результат операции | Операнды команды |

Если это прыжок, то нулевой бит результата операции (res[0] = to\_write[32]) служит сигналом должен ли был произойти прыжок.

**Архитектура системы команд (АСК)**

Команда вместе с операндами занимает 40 бит

|  |  |  |
| --- | --- | --- |
| 39..32 | 31..16 | 15..0 |
| Команда | Операнд 1 | Операнд 2 |

Биты команды имеют следующий формат

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3..0 |
| Команда является прыжком | 2 операнда | 1 операнд | 0 операндов | Номер команды |

Полный список реализуемых команд

|  |  |  |  |
| --- | --- | --- | --- |
| Обозначение | Битовое представление | HEX представление | DEC представление |
| NOTZ | 00100001 | 21 | 33 |
| AND | 01000010 | 42 | 66 |
| INCS | 00100011 | 23 | 35 |
| ROL | 00100100 | 24 | 36 |
| MOV | 01000101 | 45 | 45 |
| JMP | 10010110 | 96 | 150 |
| PUSH | 00100111 | 27 | 39 |
| POP | 00011000 | 18 | 24 |
| HLT | 00011001 | 19 | 25 |
| JZ | 10011010 | 9A | 154 |

**Извлечение операндов (OF)**

В блоке извлечения операндов находится два регистра, отвечающие за временное сохранение операндов.

Полная программа блока включает в себя загрузку обоих операндов. Управление количеством загружаемых операндов происходит за счет изменения длины выполняемой микропрограммы (константа длины для внутреннего итератора).

Даже если длина программы равна 0, то на выходе ready имеется мгновенный прыжок (101), который позволяет использовать этот выход для запуска следующего блока выполнения.

Блок полностью полагается на верную кодировку старших четырех бит команды. Если ни один бит, отвечающий за количество аргументов, не выставлен, то блок войдет в бесконечный цикл.

Также происходит проверка не является ли команда командой PUSH. Если это команда PUSH, то блок записи выключается посредством выключения бита 63 регистров WRITE\_IN.

Внутренняя шина блока (ICB[15..8])

|  |  |
| --- | --- |
| Номер бита | Значение |
| 8 | Выставить на шину адреса адрес OP1 |
| 9 | Выставить на шину адреса адрес OP2 |
| 10 | Сохранить OP1 |
| 11 | Сохранить OP2 |

Микропрограмма извлечения операндов

|  |  |  |
| --- | --- | --- |
| ICB | CB | AB |
| Ничего не делаем | | |
| 00000000\_11000000 | 00000000\_00000000 | 00000000\_00000000 |
| Выставляем адрес первого операнда на шину и защелкиваем данные | | |
| 00000001\_10110001 | 00000000\_00000000 | 00000000\_00000000 |
| Читаем первый операнд и сохраняем его | | |
| 00000101\_10110001 | 00000000\_00000000 | 00000000\_00000000 |
| Читаем второй операнд | | |
| 00000010\_10110001 | 00000000\_00000000 | 00000000\_00000000 |
| Читаем второй операнд и сохраняем его | | |
| 00001010\_10110001 | 00000000\_00000000 | 00000000\_00000000 |

**Запись результата (WRITE)**

Внутренняя шина блока (ICB[15..8])

|  |  |
| --- | --- |
| Номер бита | Значение |
| 8 | 0 – выставить значение 1- выставить флаги |
| 9 | 0 – не выставлять 1 – выставить адрес результата |
| 10 | Всегда ноль чтобы работало (tristate bustri) |

Микропрограмма

|  |  |  |
| --- | --- | --- |
| ICB | CB | AB |
| Ничего не делаем | | |
| 00000000\_11000000 | 00000000\_00000000 | 00000000\_00000000 |
| Записываем регистр флагов и защелкиваем данные | | |
| 10000001\_11111000 | 00000000\_00000001 | 00100000\_00000000 |
| Выставляем адрес результата и записываем данные | | |
| 00000010\_10111000 | 00000000\_00000001 | 00100000\_00000000 |
| 00000010\_10111000 | 00000000\_00000001 | 00100000\_00000000 |

**Регистр флагов (FR)**

Имеет адрес 0 в пространстве регистров.

Значащим является только младший байт. Значения старшего байта не определены.

|  |  |
| --- | --- |
| Бит | Значение |
| 0 | 0 – (res != 0) 1 – (res == 0) |
| 1 | 0 – (res > 0) 1 – (res < 0) |
| 2 | Carry бит |
| 3 | Бит переполнения |
| 4 | Стек пуст ошибка чтения |
| 5 | Стек полон ошибка записи |
| 6 |  |
| 7 |  |

**Конвейер (convey)**

Конвейер управляется извне контрольным модулем. Конвейер имеет три этапа выполнения:

|  |  |
| --- | --- |
| этапа выполнения | значение |
| IF | Извлечение команды |
| OF & EXEC | Извлечение операндов и выполнение операции |
| WRITE | Запись результатов |

0 и 1 фазы выделены для работы всех блоков т.к. параллельно работают блоки (фаза 0 - IF || OF) а потом (фаза 1 - EXEC || WRITE).

Конвейер работает постоянно. Остановка происходит за счет своевременной установки битов reg[63] в 0.

**Контрольный модуль (CM)**

Осуществляет контроль конвейера.

Имеет следующие этапы:

|  |  |
| --- | --- |
| Этап | Значение |
| 0 | Запустить все блоки конвейера, если нет конфликтов чтения-записи |
| 1 | Запустить блок записи, если есть конфликт чтения записи(добавить смену регистров записи и выключение) |
| 2 | Если предыдущий прыжок был с неверным предсказанием, то исправить адрес и сохранить флаг «неверный прыжок» |
| 3 | Перезаписать промежуточные регистры если нет конфликтов чтения-записи |
| 4 | Сменить промежуточные регистры если нет конфликтов чтения-записи  (вторая половина такта) Остановить блок WRITE если есть конфликт чтения-записи |
| 5 | Остановить блок EXEC если выставлен флаг «неверный прыжок»  Перезаписать текущее значение конфликта чтения-записи |

**Диспетчер прямого доступа к памяти (DMA)**

Последовательный централизованный кпдп

Реализация монтажного или невозможна доступными средствами. Поэтому пришлось использовать двухлинейное соединение с каждым ведущим устройством (шина DMAB[5..0]).

Описание шины DMAB[5..0]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 | 0 |
| Занятость | | | Запрос | | |
| EXTERNAL | WRITE | OF | EXTERNAL | WRITE | OF |

Каждый ведущий блок имеет внутри себя блок DMA\_check. Этот блок осуществляет получение доступа к шине и только потом позволяет работать ведущему устройству.

Разрешенный тактовый сигнал (есть только когда блок получил управление) используется во всем блоке. Однако тактовый сигнал для rom, который содержит внутренние микрокоманды, должен быть всегда т.к. хоть адрес микрокоманды и сменился, rom требует дополнительного такта для переключения текущей микрокоманды на первую при завершении работы блока.

**Контроллер прямого доступа к памяти (DMA\_worker)**

Реализует блочное копирование данных.

Имеет внутри себя настройки: source\_address, destination\_address, words\_count, enabled. Выставление enabled в 1 запускает работу блока.

|  |  |
| --- | --- |
| Линия шины | Значение |
| 8 | Выставить адрес и данные на \_AB \_DB |

Микропрограмма DMA\_worker:

|  |  |  |
| --- | --- | --- |
| ICB | CB | AB |
| Ничего не делаем | | |
| 00000000\_11000000 | 00000000\_00000000 | 00000000\_00000000 |
| Выставить адрес и данные на \_AB \_DB -> AB DB CB и записать | | |
| 00000000\_10111000 | 00000000\_00000001 | 00000000\_00000000 |

Контроллер имеет внутри себя блок настроек, который включает в себя следующие регистры:

|  |  |
| --- | --- |
| Адрес(hex) | Значение |
| 3000 | Количество слов, которые нужно передать |
| 3001 | Адрес назначения |
| 3002 | Адрес источника |
| 3003 | Вкл (все биты 1)/выкл (хотябы один бит 0) |

При выставлении адреса такого регистра автоматически подразумевается запись нового значения и младший бит CB не имеет никакого значения.

**Операции**

Каждая операция имеет отдельную линию для запуска. Эта линия остается активной в течение одного такта конвейера. Все выходы блока должны быть проведены через tri с условием, что блок включен для избежания конфликов результата и флагов.

**AND (DEC 66)**

|  |  |  |
| --- | --- | --- |
| ICB | CB | AB |
| Ничего не делаем | | |
| 00000000\_11000000 | 00000000\_00000000 | 00000000\_00000000 |
| Защелкиваем входные данные | | |
| 10000000\_11000000 | 00000000\_00000001 | 00100000\_00000000 |

**HLT (DEC 25)**

Операция HLT срабатывает мгновенно. Сам блок представляет собой выключенный регистр состояния. При старте он обнуляется. Если запускается команда HLT, то регистр устанавливается в состояние 1. Это состояние передается по шине CB\_convey[6] и обозначает, что необходимо остановить устройство в конце выполнения текущего цикла обработки (чтобы успел отработать блок записи).

**PUSH (DEC 39)**

Кладет значение первого операнда в стек. После этой операции необходима исключительно запись регистра результата.

Старшая часть шины ICB[15..8] отвечает за управление блоком. Младшая часть шины ICB[7..0] выставляется на шину CB\_stack[7..0] и отвечает за управление стеком.

Значение старшего байта шины ICB[15..8]:

|  |  |
| --- | --- |
| Номер байта ICB[15..8] | Значение |
| 8 | Выставить операнд на шину данных стека DB\_stack[15..0] |
| 9 | Защелкнуть значение флагов и результата |

Микропрограмма

|  |  |
| --- | --- |
| Значение шины ICB[15..0] | Значение |
| 00000001\_00001000 | Записать операнд на текущее место указателя стека x2 |
| 00000000\_00000010 | Увеличить текущее значение указателя стека |
| 00000010\_00100000 | Прочитать значение флагов и защелкнуть его |

**POP (DEC 24)**

Достает последнее значение из стека.

Старшая часть шины ICB[15..8] отвечает за управление блоком. Младшая часть шины ICB[7..0] выставляется на шину CB\_stack[7..0] и отвечает за управление стеком.

Значение старшего байта шины ICB[15..8]:

|  |  |
| --- | --- |
| Номер байта ICB[15..8] | Значение |
| 8 | Защелкнуть значение результата |
| 9 | Защелкнуть значение флагов |

Микропрограмма

|  |  |
| --- | --- |
| Значение шины ICB[15..0] | Значение |
| 00000000\_00000100 | Уменьшить текущее значение указателя стека на единицу |
| 00000001\_00010000 | Прочитать верхнее значение стека х2 и защелкнуть его |
| 00000010\_00100000 | Прочитать значение флагов и защелкнуть его |

**MOV**

Переносит второй операнд на место первого.

**Тестовая программа**

|  |  |  |
| --- | --- | --- |
| NOTZ | 00100001 | 21 |
| AND | 01000010 | 42 |
| INCS | 00100011 | 23 |
| ROL | 00100100 | 24 |
| MOV | 01000101 | 45 |
| JMP | 10010110 | 96 |
| PUSH | 00100111 | 27 |
| POP | 00011000 | 18 |
| HLT | 00011001 | 19 |
| JZ | 10011010 | 9A |

Заполним RAM 0 1 2 3 4..

|  |  |  |  |
| --- | --- | --- | --- |
| № | Команда | Значение | Численные значения(hex) |
| 0 | 4530001030 | Колво слов RAM[30] | 10 |
| 1 | 4530011031 | Адрес назначения RAM[31] | 1020 |
| 2 | 4530021032 | Адрес источника RAM[32] | 0040 |
| 3 | 4530031033 | Включить все в 1 RAM[33] | FFFF |
| 4 | 4220011001 | RCM[1] & RAM[1] -> RCM[1] | 0 & 1 -> 0 |
| 5 | 2110030000 | !RAM[3] -> RAM[3] | !3 |
| 6 | 2110030000 | RAM[3] -> RAM[3] | !3 |
| 7 | 2310000000 | RAM[0] + 1 -> RAM[0] | 0 + 1 -> 1 |
| 8 | 2310000000 | RAM[0] + 0 -> RAM[0] | 1 + 0 -> 1 |
| 9 | 2410020000 | RAM[2] << 1 -> RAM[2] | 2 << 1 -> 4 |
| 10 | 4520031004 | RAM[4] -> RCM[3] | 4 -> 4 |
| 11 | 4510052003 | RCM[3] -> RAM[5] | 4 -> 4 |
| 12 | 2710100000 | RAM[10] -> STACK | 10 -> 10 |
| 13 | 1810070000 | STACK -> RAM[7] | 10 -> 10 |
| 14 | 1900000000 | HLT |  |
| 15 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |