**Построение собственной архитектуры ЭВМ.**

Книга от среднестатистического ученика 11 класса.

**В гостях у “сказки”**

**о науке и логике**

**через испытания к**

**“великим” знаниям**

**Содержание**

|  |  |
| --- | --- |
| Свойства проекта | 3 |
| Вступление | 4 |
| Теоретическая часть | 5 |
| Логические вентили | 6 |
| Комбинационные схемы | 8 |
| Архитектуры | 9 |
| Структура ЭВМ | 11 |
| Структура Процессора | 12 |
| АЛУ | 13 |
| Единая структура | 14 |
| Модульная структура | 15 |
| Регистры | 16 |
| Устройство Управления | 17 |
| Устройство декодирования команд | 18 |
| Устройство тактирования | 19 |
| Практическая часть | 20 |
| Идея | 21 |
| Список команд | 23 |
| Подготовка | 24 |
| Реализация | 30 |
| Проектирование УУ |  |
| Тип команд 1 |  |
| Тип команд 2 |  |
| Тип команд 3 |  |
| Тип команд 4 |  |
| Итог |  |
|  |  |

**Свойства проекта**

**Цель:**

|  |
| --- |
| * Собрать собственную архитектуру процессора. (продукт проекта 1) |
| * Рассказать о логических схемах в формате книги. (продукт проекта 2) |

**Задачи:**

|  |
| --- |
| * Найти информацию об архитектурах. |
| * Найти похожие проекты и рассказать о них. |
| * Рассказать факты, касающиеся данной темы, для большего понимания. |
| * Собрать информацию о структуре компонентов и их разновидностях в ЭВМ. |
| * Предложить собственные идеи о структуре компонентов и их разновидностях в ЭВМ. |
| * Составить набор выполняемых команд собственной архитектуры. |
| * Составить логическую схему собственной архитектуры. |

**Актуальность:**

|  |
| --- |
| * Малое количество простых индивидуальных архитектур, как процессоров, так и ЭВМ. |
| * Малое количество свободной для понимания информации в Интернете. |

**Теоретическая значимость:**

|  |
| --- |
| * Люди смогут узнать и разобраться в устройстве многих компонентов ЭВМ, и далее применить знания на практике (при соответствующем желании). |

**Практическая значимость:**

|  |
| --- |
| * Практическая часть даст возможность досконального изучения и выявления некоторых нюансов при создании ЭВМ. |

**Объект исследования:**

|  |
| --- |
| * Архитектура ЭВМ |

**Предмет исследования:**

|  |
| --- |
| * Конструирование ЭВМ |

**Вступление**

Здравствуйте, данная книга является продуктом (и собственно копией) индивидуального проекта учащегося 11 класса. Тема, которую я затрагиваю очень редка, и мало кому интересна. Наука изучающая Логические схемы забывается среди населения из-за перехода в промышленность. Крупные компании монополизируют рынок микросхем, а малые производства, не имея многих знаний в данной сфере, практически не способны развиваться. Самые основные устройства и их схемы в интернете, конечно, доступны, но разнообразия почти что нет. Это я имел в виду лишь современную информацию, а литература советского времени довольно богата знаниями. Что говорит об остановке развития науки среди доступного населению пространства. Основные исследования проводят крупные компании, они патентуют почти всё, ведь в ином случае произойдёт потеря прибыли. На момент 2020г. компании ведут яростную борьбу за потребителя, так будет всегда, поэтому в доходах они особо заинтересованы. По данным причинам энтузиасты, желающие сделать что-то и зачем-то, не могут проникнуть в данную сферу. А ведь таких людей не мало, и я сам знаю 6 человек (правда они меня не знают). Так что я вовлечённый в данную научную сферу решил написать данный проект.

Проект рассказывает о создании собственной простейшей архитектуры ЭВМ. Современные ЭВМ как правило имеют процессор, то есть я не буду рассказывать о создании просто процессора, я решил замахнуться и спроектировать ЭВМ целиком. Как и вам полезно, можете прочитать об интересующем вас элементе, так и мне есть чем заняться.

**Теоретическая часть**

Перед началом собственно проектирования нужно понимать, что мы собрались проектировать.

ЭВМ – Электронно-Вычислительная Машина, выполняет вычисления используя электроэнергию. Вы ведь помните, что такое ламповый компьютер, первейший класс ЭВМ? Тогда и названия “компьютер” не было. Они выполняли различные логические и математические задачи. Потребляли много энергии и были размером в 1-4 комнат. А потом с изобретения Транзистора началась новая эра.

Транзистор был отличной заменой Вакуумным электронным лампам. Потребления энергии меньше, размеры намного меньше, способ эксплуатации легче и надёжность выше. Потом люди додумались делать несколько транзисторов в одном корпусе, придумали литографию и начали выпускать первые микросхемы, а там и до процессоров не далеко было.

И да! Процессор как термин появился позже ЭВМ. Но у многих ЭВМ структура позволяла наличие Процессора, как главного вычислительного блока.

Теперь по существу…

Что нам надо знать для проектирования собственной ЭВМ? На самом деле, очень много. Начиная с логических вентилей, заканчивая устройством адресации доп. устройств.

**Логические вентили**

Логические вентили (Логические элементы ЛЭ) - базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Обозначение | | Описание |
| **Международное** | **IEC 60617-12 : 1997** |
| NOT  (НЕ) |  |  | Инвертирует входящий сигнал. |
| AND  (И) |  |  | Выдаёт сигнал только при наличии сигнала на всех своих входах. |
| OR  (ИЛИ) |  |  | Выдаёт сигнал только при наличии сигнала на любом из своих входов. |
| XOR  (Исключающее ИЛИ) |  |  | Выдаёт сигнал только при наличии сигнала на одном любом из своих входов. |
| NAND  (И-НЕ) |  |  | Выдаёт сигнал только при отсутствии сигнала на хотя бы одном из всех своих входов. |
| NOR  (ИЛИ-НЕ) |  |  | Выдаёт сигнал только при отсутствии сигнала на всех своих входах. |
| XNOR  (Исключающее ИЛИ-НЕ) |  |  | Не выдаёт сигнал только при наличии сигнала на одном любом из своих входов. |

А проще говоря это, элемент выполняющий какую-то простейшую логику основываясь на входящих сигналах. Они могут иметь как 1 вход, так и 8, или более. И могут состоять из Электронных ламп, транзисторов или чего-нибудь ещё что будет сказываться на их характеристиках. Но всегда имеют 1 выход. Для двоичной логики Логических вентилей существует ровно 7.

Если количество выходных сигналов больше одного, то Логическим элементом это назвать нельзя. Такой элемент уже будет являться Логическим модулем, состоящим из Логических элементов.

|  |  |
| --- | --- |
| Название | Таблица истинности |
| NOT  (НЕ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | - | 1 | | 1 | - | 0 | | - | - | - | | - | - | - | |  |  |  | |
| AND  (И) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 0 | | 1 | 0 | 0 | | 0 | 1 | 0 | | 1 | 1 | 1 | |  |  |  | |
| OR  (ИЛИ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 0 | | 1 | 0 | 1 | | 0 | 1 | 1 | | 1 | 1 | 1 | |  |  |  | |
| XOR  (Исключающее ИЛИ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 0 | | 1 | 0 | 1 | | 0 | 1 | 1 | | 1 | 1 | 0 | |  |  |  | |
| NAND  (И-НЕ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 1 | | 1 | 0 | 1 | | 0 | 1 | 1 | | 1 | 1 | 0 | |  |  |  | |
| NOR  (ИЛИ-НЕ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 1 | | 1 | 0 | 0 | | 0 | 1 | 0 | | 1 | 1 | 0 | |  |  |  | |
| XNOR  (Исключающее ИЛИ-НЕ) | |  |  |  | | --- | --- | --- | |  |  |  | | A | **B** | **Out** | | 0 | 0 | 1 | | 1 | 0 | 0 | | 0 | 1 | 0 | | 1 | 1 | 1 | |

Логику работы этих Логических элементов можно записать в таблицу, которая называется Таблица истинности. Такие таблицы могут быть практически бесконечны! Смысл в том, что они дают понять логику работы и возможность взглянуть на схему, с другой стороны.

Также стоит запомнить и научиться составлять Логические формулы. Чем-то они похожи на физические или математические формулы, но здесь используются определённые символы (которые вам тоже стоит понимать). С помощью этих формул можно посмотреть на логику работы с математической стороны, что может помочь с оптимизацией схемы.

|  |  |  |
| --- | --- | --- |
| Название | Обозначение | Описание |
| Инверсия |  | Инвертирует значение.  Соответствует вентилю НЕ. |
| Конъюнкция | , | Умножает значения.  Соответствует вентилю И. |
| Дизъюнкция | , | Складывает значения.  Соответствует вентилю ИЛИ. |
| Эквиваленция | , | Сравнивает значения.  Соответствует вентилю Исключающее ИЛИ-НЕ. |
| Импликация |  | Если A то возвращается B, иначе 1.  Соответствует вентилю И. |

Если вы уже создали свою Логическую схему и хотите её реализовать в жизнь, начните с оптимизации Логической схемы. При оптимизации необходимо учитывать радиоэлектронную схему Логических вентилей, их структуру. Например, Элемент И сложнее чем элемент И-НЕ, разница всего на 1-3 Транзистора вам принесёт много проблем с реализацией вашей схемы. Конечно, это зависит от технологии Логического вентиля (ТТЛ, МОП, КМОП, и т.д.). Но оптимизация никому пока не мешала, а в индустрии микроэлектроники это, очень важная задача от которой, можно сказать, зависит всё.

**Комбинационные схемы**

Это элементы, представляющие собой совокупность Логических Элементов. Они являются прямым продолжением вентилей, могут иметь любую логику работы и любое количество входов и выходов.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ЛогическаяСхемаМЕгоПредставлениеВВидеЛЭ.PNG |
| Логическая схема и её представление в виде элемента |

Эти схемы сочетают в себе два вида логики. Собственно, Комбинационную, где входные сигналы прямо влияют на выходные, и Секвенциальная, которая для формирования выходных сигналов использует внутри себя триггеры (схему способную запоминать своё состояние).

Обычно, насчёт названия таких схем точной информации не говорится. Где-то это «схемы», а где-то «Комбинационные схемы» или «логические схемы». Но как по мне, здесь верным будет именно «Комбинационные элементы», так как оно передаёт точный смысл и ни где не используется более для обозначения чего-либо другого.

**Архитектуры**

Архитектура, это довольно общее понятие, собирающее в себе много определённых терминов. Она описывает как структуру с функциями, так и характеристики. Легче будет усвоить информацию разбирая отдельные классы Архитектур, которые соответствуют нашей теме.

Начнём с Архитектуры ЭВМ. Коих имеется много, но основными являются всего два. Это Архитектура Фон-Неймана и Архитектура Гарварда.

|  |  |  |
| --- | --- | --- |
| Фон Нейман |  | Гарвард |
| |  | | --- | | Общая  память | |  | | Процессор | |  | | Прочие  устройства | | |  |  |  | | --- | --- | --- | | Память  данных |  | Память  команд | |  |  |  | | Процессор | | | |  | | | | Прочие  устройства | | | |

|  |  |  |
| --- | --- | --- |
| Класс | Описание | Архитектуры |
| RISC | **Reduced Instruction Set Computer**  С Сокращенным Набором Команд Компьютера. Особенностью данной группы является увеличение производительности за счёт сокращения набора команд, редко используемых разработчиками ПО. Является золотой серединой. | ARM, MIPS, OpenRISC, RISC-V и др. |
| TTA | **Transport Triggered Architecture**  Транспортно-вызываемая архитектура. Архитектура на основе всего одной инструкции перемещения из одного адреса памяти в другую. Очень производительная архитектура, которая расплачивается сложностью программирования. | MOVE Project и др. |
| OISC | **One Instruction Set Computer**  Архитектура с единственной инструкцией. Такие архитектуры часто имеют вид: Сделать действие и в зависимости от результата сделать прыжок или продолжить исполнение. Зачастую ее реализация достаточно простая, производительность низкая. | BitBitJump, ByteByteJump, SUBLEQ и др. |
| CISC | **Complex Instruction Set Computer**  Компьютер Со Сложным Набором Команд. Ее особенность в увеличенном количестве действий за инструкцию. Таким образом можно было теоретически увеличить производительность программ за счет увеличения сложности компилятора. Огромное множество архитектур этой группы почти не имеют схожести между собой. | x86, x86-64, M68K и др. |

Обе этих архитектур совершенно различны и имеют разные задачи. Например, Архитектура Фон-Неймана позволяет написать и запустить программу на уровне машинного кода (Ассемблер), что мы и встречаем в современных компьютерах, а Архитектура Гарварда используется в устройствах где всегда будет выполнятся одна и та же программа, как в микроконтроллерах. Так же стоит отметить, что Архитектура Фон-Неймана уступает по производительности Архитектуре Гарварда, из-за ограничений на запись и чтение данных. Прирост составляет 20% при использовании одноканальной памяти.

Далее идёт Архитектура Процессора. Это совокупность главных принципов его конструирования, общая схема расположения деталей и схема взаимодействия программного обеспечения с чипом. И данных Архитектур огромное множество, что были созданы различные классификации. Такие как: RISC, TTA, OISC, CISC, итд. А уже, например, Zen3, M68k, x86 и IceLake это непосредственно Архитектуры, которые мы детально рассмотреть увы не сможем.

Как видно из таблицы Классов Архитектур, классифицируются они по командам выполняемым процессором, или же по их Ассемблеру. Но различные модули процессора и их строение, которое нам сильно понадобятся, относятся непосредственно к Архитектурам Процессора, и классификации не принадлежат (из-за разных реализаций разными производителями).

(Например, никто не знает логическую схему модуля предсказания ветвлений, который выполняет заранее более ожидаемые, после предыдущей, команды)

Конечно в Интернете есть схемы мелких модулей. И находятся они в оцифрованных патентах СССР. Как устройство двоичного деления, в схеме которого я не разобрался. И спроектировал своё устройство деления. Но об этом поговорим в будущем.

**Структура ЭВМ**

Как и всё в этом мире, ЭВМ тоже состоит из чего-то. И это компоненты в виде различных модулей или устройств.

Для выполнения списка команд ЭВМ имеет Память, как скоростную, так и постоянную. Отличие в том, что энергонезависимую память сделать быстрой очень проблематично. Современные SSD накопители так же не удовлетворяют нашим скоростным потребностям. И тут без скоростной энергозависимой Оперативной памяти не обойтись. Она состоит из тех же компонентов что и Процессор и способна работать чуть медленнее самого Процессора. Поэтому, в любом универсальном вычислительном устройстве есть как Оперативная память, так и Постоянная. ОЗУ и ПЗУ соответственно.

Далее любому устройству, что рассчитан на возможность использования Интерфейсов, требуются Устройства Ввода и/или Устройства Вывода информации. Допустим, калькулятор не поймёт, что считать если вы не передали ему что считать. А вы в свою очередь не поймёте, что получилось в итоге если калькулятор толком не может никак вывести информацию. Даже 1 кнопка или просто антенна из оголённого провода является Устройствами Ввода, как и обычный электромотор, нить накала или обычный светодиод являются Устройствами Вывода. Но в нормальных условиях Клавиатура или TouchPad – Устройства Ввода, а дисплей Устройство Вывода.

Наверное, самый главный элемент любого ЭВМ, это Процессор, но он не обязателен. Например, в ЭВМ 1970 года не было Процессора. Ни как элемент схемы, и ни как устройство по современным определениям. И элементы, которые, казалось бы, есть только в процессоре, вплетены в саму Архитектуру ЭВМ.

Далее о таких элементах как, устройство тактирования, Устройство декодирования команд, Блок питания, и т.д. рассказывается здесь не будет. Некоторые относятся к другому параграфу, или в достижении нашей цели роли не играют.

**Структура Процессора**

Процессор основное вычислительное устройство всех современных ЭВМ. От вашего холодильника до Квантовых компьютеров. Данное устройство имеет блочную структуру своих модулей. Некоторые из модулей Процессора являются необязательными, но наличие каждого из модулей сильно отражается на Архитектуре Процессора.

Самыми основными и базовыми модулями являются: Управляющее Устройство, Арифметико-Логическое Устройство, Регистры.

Второстепенными модулями являются: Ядро Процессора, Стек, Устройство Математических Операций, Устройство Блокировки Доступа или Устройство Уровней Доступа, Устройства Предсказания Ветвлений, Кеш, Устройство Шифрования, Устройство Аппаратного Декодирования, Графический Модуль и т.д..

Различных компонентов Процессора очень много. И вы, наверное, обратили своё внимание на Ядро. Вроде в каждом процессоре есть хотя бы одно ядро, но вспомните первые Процессоры, где Ядер как отдельного модуля не было. Компоненты современных ядер были вплетены в Архитектуру Процессора, как это и было с Процессором и ЭВМ.

Стоит сказать и о способах объединения модулей между собой. Шина. Не является устройством и состоит только из проводников, являясь проводами, соединяющими все компоненты. Но шина в отличие просто от проводов, является группой проводников передающие одинаковую информацию всем подключенным к ней модулям. Кроме того, шин может быть несколько.

**АЛУ**

Арифметико-Логическое Устройство, основной компонент любой ЭВМ. Выполняющий базовые математические и логические операции способен реализовать любую задачу, и вычисление числа Pi одним лишь АЛУ возможно.

У АЛУ присутствуют и свои Архитектуры, так производители могут в разные Архитектуры Процессора встроить нужный им АЛУ. На практике же АЛУ жёстко привязана к Архитектуре Процессора, и дорабатывается совместно с ней.

Подключается данный модуль Процессора к шине или шинам через регистры. И в качестве некоторых исключений разрядность АЛУ и шины не совпадают. Помимо шины, АЛУ подключается к УУ (Устройство Управления) через 2х-6х разрядную шину определяющую требуемую операцию.

Некоторые виды АЛУ имеют модули умножения и деления чисел, что редкость среди индивидуальных архитектур, а в современных процессорах архитектуры x86 эти модули входят в состав множества пакетов расширения, например SSE, предназначенного для расширения математических способностей Процессора.

|  |
| --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/1/16/ALU_block-ru.svg/1280px-ALU_block-ru.svg.png |
| Обобщённая блок-схема АЛУ |

Самое простое АЛУ имеет 2 входа, причём 1 из них постоянно связан через регистр с выходом этой же АЛУ. Имеет входы для Флагов и Команды. И естественно выходные Флаги.  
(Флаг - это проводник, предназначенный для передачи какой-либо информации)

По совокупности команд АЛУ обычно не различают. Они могут выполнять от двух до сотен операций. Но вот вопрос в другом. Как быстро они выполняют такое множество операций? Ведь простейших операций очень мало, значит остальные являются очень сложными! Тут АЛУ решают данную задачу в несколько тактов! Такой подход позволяет выполнять в сколь угодно сложные операции путём расширения блока управления АЛУ. Это очень сильно изменяет её архитектуру и данные АЛУ в большинстве имеют модульный тип.

**АЛУ, Единая структура**

Некоторые АЛУ могут представлять из себя 1 большую схему. Такой подход экономит количество Логических Элементов. Но данные АЛУ Очень сложны в разработке! Обычно они выполняют несколько операций, выполняя каждую из них в 1 такт. Также они не могут комбинировать действия, что плохо сказывается на итоговой производительности. Сейчас такие АЛУ используются исключительно как дополнительные математические блоки в архитектурах многих элементов современного процессора. Такие АЛУ даже используются в блоке управления основных сложных модульных АЛУ.

|  |
| --- |
| https://upload.wikimedia.org/wikipedia/commons/c/c0/74181aluschematic.png |
| Комбинационная логическая схема 4-битного АЛУ, реализованная в 24-выводной микросхеме ТТЛ, модель 74181 |

Можно наглядно увидеть, что АЛУ реализована как 1 устройство. Видите, как схема слева на право усложняется? Это и есть основная причина сложности таких АЛУ, ведь на каждый новый бит влияют все предыдущие. Такие АЛУ имеют модульные схемы, только если эти схемы не требуют связи между собой. Все устройства в них перемешаны и некоторые делали выполняют порой 3 или более функций.

Такие монолитно единые схемы проектируют с помощью Логических Формул. Эти Формулы объединяют изменяют дополняют и, что самое главное, их сокращают! Благодаря сокращениям этих формул, по неким законам, мы получаем экономию логических элементов, но мы не сможем интуитивно понять смысл выражения.

**АЛУ, Модульная структура**

АЛУ Модульного типа чрезвычайно распространены. Не только за счёт своей лёгкости, но и за счёт возможности выполнения нескольких действий! Число проходя последовательно через каждый модуль, может быть обработано и изменено любым из них. Таким образом Модульные АЛУ могут произвести за 1 такт следующее вычисление: (А+В)&В.

|  |
| --- |
|  |
| Комбинационная схема АЛУ (пример) |

В данном примере вы можете чётко заметить длинную последовательность из различных комбинационных схем. Сразу видно за что отвечают управляющие Флаги, не смотря на их название. Здесь можно с лёгкостью вычесть из B число A и произвести булево умножение результата с числом B. Для этого понадобится активировать Swap, isSub, isAnd и isGet. Такого простора к комбинации операций нам АЛУ с единой структурой дать не могла. Но и этот тип АЛУ не очень хорош. Здесь очень много логических элементов что плохо сказывается на огромном множестве факторов, которые встанут перед реализацией этой схемы в жизни.

Этот тип АЛУ используется в Процессорах как основной, так как другой альтернативы не предвещается. Но уже на данный момент в производители совмещают некоторые модули АЛУ и оптимизируют их.

**Регистры**

Это очень гибкие в управлении устройства временного хранения информации. В Процессорах регистры являются основополагающим элементом производительности. Даже Кеш в современных процессорах можно назвать Регистром.

Любой регистр должен уметь запоминать и выводить информацию по команде. К тому же, записывать информацию они могут как при появлении управляющего сигнала, так и при его выключении (фронт и спад соответственно).

Иногда в отличии от обычных данных, они хранят флаги. На основе либо информации, либо Флагов, либо всего вместе Устройство Управления выдаёт определённые команды остальным компонентам процессора.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Регистры.png |
| Структура современного регистра данных, где AH и AL имеют 8 бит. |

Многие много битные Регистры на просто состоят из нескольких меньшей битности, и во многих процессорах эти регистры можно использовать отдельно. Это всегда выручало первые процессоры, так как их битность была даже меньше 8 бит, то и обратиться теоретически они могли лишь к 256 ячейкам памяти, что очень мало. Парные Регистры по сути могли хранить 8х2 бит данных, а значит и обращаться через этот регистр процессор мог уже к 65535 ячейкам памяти. (если ячейка памяти это 8 бит, то это равно 64Кб)

Многие процессоры могут эмулировать работу в двое большей битности за счёт замедления вычислений в 2 раза. Происходит это из-за того, что в первый такт считается первая часть числа, после сохраняются все флаги, и на основе флагов в второй такт считается вторая часть числа. Даже существуют процессоры, которые всегда работают в таком режиме.

**УУ**

Устройство Управления процессора – это устройство формирующее управляющие сигналы всех компонентов процессора, основываясь на совокупности состояний компонентов этого же процессора.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\УУ.PNG |
| Простейшая схема процессора |

Самый Главный и большой блок любого процессора! Определяет логику работы всех компонентов. Имеет очень сложную структуру, частично модульную.

Первое что делает УУ – это приём первой команды. УУ всегда формирует после запуска необходимые сигналы на подгрузку из ОЗУ первой команды. После уже команда выполняется и подгружаются следующие.

Принимая команду, УУ должна превратить её в управляющие сигналы. Обычно бля этого команды делятся на типы, которые делятся на подтипы, которые уже делятся на разновидности команд. Таким образом при проектировании экономится огромное количество сил, ведь проектировать придётся не единую систему, реагирующую на всё сразу, а модульную систему где каждая часть выполняет строго свои действия по обработке команд и флагов, не выполняя ничего лишнего.

После дешифровки команды у УУ готовы управляющие сигналы. Они наспех не отправляются к компонентам процессора. В первую очередь УУ проверяет возможность отложение команды на след такт и на возможность приёма дополнительных данных. Только после всех этих процедур УУ выдаёт управляющие сигналы к компонентам процессора.

Но если вы считаете, что УУ полностью контролирует все компоненты, то вы ошибаетесь. Есть в Процессоре компоненты, которые могут работать самостоятельно и без каких-либо входящих данных. К таким устройствам относится, например, Счётчики. Они и без команд от УУ считают каждый такт, и УУ только и остаётся, что считывать значение или же обнулять их.

**УУ, Устройство декодирования команд**

Самый сложный элемент любого процессора! Но при грамотном проектировании это может оказаться и вовсе не сложный компонент процессора. И дело тут не в умении людей всё упрощать, а в природе логики.

История УУ тянется от создания программируемых ЭВМ. С тех пор люди научились вкладывать закономерные команды в проекты УУ. Эти закономерности напрямую используются в Любом устройстве декодирования команд!

Пример. У нас есть 4 регистра и 4 команды, которые делают одно и то же для отдельного регистра. Почему бы нам этим не воспользоваться? Переводим команды в двоичный вид и в первые 2 разряда закладываем номер регистра, с которым надо произвести операцию. После в Устройстве декодирования на эти младшие 2 разряда устанавливаем декодер. (устройство, которое выводит напряжение на тот контакт, чей номер указан на входе). Теперь мы можем сделать проверку на команду, а далее младшие 2 разряда укажут на номер регистра, с которым надо работать.

Такие закономерности специально создаются для упрощения Устройства декодирования. Но это устройство создаёт лишь первичные управляющие сигналы. Далее в самом УУ эти сигналы подвергнутся дополнительной обработке. Там уже закономерности сложнее предусмотреть.

После Устройства декодирования идёт сложная система сохранения, чтения, анализа и игнорирования тактов. По сути это можно назвать системой контроля тактов. Устройство декодирования само вместе со всеми управляющими сигналами выдаёт ему команды на контроль тактов. Нужно это всё из-за сложности некоторых команд. Они могут выполнятся, подгружать аргументы команды или данные в несколько тактов.

**УУ, Устройство тактирования**

Наверное, самое простое в плане объяснения устройство, но в то же время не менее важное. Основная задача этого устройства – создание необходимых переменных импульсов.

|  |
| --- |
|  |
| Преобразователь синусоидального сигнала в прямоугольный |

Для реализации этого устройства в основном используют колебательный контур. Но контур производит синусоидальный сигнал, а в логических схемах средние напряжения крайне не приветствуются. Поэтому, после колебательного контура стоит операционный усилитель.

В итоге мы получили прямоугольный сигнал, и к сожалению, или к счастью, этого нам может не хватить. Некоторые устройства могут принимать вплоть до 10 различных тактовых сигналов! И каждый из них – индивидуальный. В основном их получают из счётчиков, чтобы не морочится с сложными схемами на конденсаторах. То есть в схемах считают каждый такт и на основе этого производят логические операции дабы получить необходимую совокупность сигналов. И вроде бы транзисторов много тратится, но это легко окупается. Ведь будь схема на конденсаторах, она бы не могла динамически менять частоты, в отличии от счётчика.

**Практическая часть**

И так, мы уже подобрались к тому, что делают инженеры в компаниях по микроэлектронике. К Проектированию!

С начала идёт подготовка всего что понадобится для работы. Записывайте:

* Электроэнергия.
* ЭВМ с ОС Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, можно и без ЭВМ.
* Обязательно либо Клавиатура, либо Мышь.
* Любой монитор от 2010 года.
* Программа для проектирования логических схем. (рекомендую Ручку и Бумагу)

А если серьёзно, то конкретно для проектирования нужно иметь компьютер и программу по проектированию логических схем. Лично я выбрал бесплатную программу «Logical Circuit». У неё простой интерфейс и нет, как в других подобных программах, электронных компонентов, что поможет не запутаться в разнообразии её функций и возможностей.

**Идея**

Для начала нужно понять, что и как проектировать для того чтобы сделать ЭВМ. Первым приходит в голову то что я описывал ранее. Это верная мысль, но не достаточная. Нам требуется общая схема.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\схема ЭВМ.PNG |
| Общая схема ЭВМ |

Любая современная ЭВМ состоит из 4х элементов. Первые 2 элемента это блоки ввода и вывода информации. Остальные 2 элемента это блоки вычислений. Так как блоки ввода и вывода кажутся простыми, можно их отложить на последний план. Возьмёмся сразу за процессор и память.

|  |
| --- |
|  |
| Схема Центрального Процессора |

Теперь нам нужна общая схема процессора. Тут мы видим, что все элементы объединяет единая шина. В некоторых реализациях шина может быть разделена через регистр. Так первая часть шины принимает или отдаёт данные, а вторая часть шины указывает номер ячейки памяти, с которой надо работать (адрес). Это общепринятая структура ЦП. А есть и иные структуры. Есть ЦП с 2 шинами данных, или с несколькими АЛУ, или с чем-либо ещё. Это всё зависит от требуемых параметров процессора. Так если вы хотите, чтобы 1 команда выполнилась за 1 такт, то нужно две шины минимум. По первой передаётся операнд А, по второй операнд Б. но в современных процессорах эта проблема решается путём такой технологии как «конвейер». Он хранит в себе элементарные задачи (микрокоманды). С каждым тактом, микрокоманда передвигается по конвейеру от последней ячейки к первой. И когда в первой ячейке появляется микрокоманда, то выполняются все микрокоманды в конвейере. Они опять сдвигаются и опять выполняются далее каждый такт, и каждый, такой, такт у процессора есть данные для следующей микрокоманды в конвейере.

Это была лишь 1 загвоздка в начале проектирования собственной Архитектуры. И таких загвоздок очень много, и связанны они почти со всеми элементами ЦП и ЭВМ. Но придётся через всё это пройти.

Так что же выбрать с этими регистрами и выполнением за 1 такт? Ведь если использовать 1 шину, то мы на подгрузку данных из регистра в АЛУ потратим 2 такта. Явно конвейеры это Очень тяжёлая технология. Будем делать 2 шины данных. Это сильно усложнит УУ. Ведь помимо управляющих сигналов регистра «Set» и «Get», придётся контролировать их же, но для каждой шины отдельно, и выходит у нас «Set1», «Get1», «Set2» и «Get2». Исходя из 4х управляющих сигналах на регистр, я считаю, что 4 регистра (А, Б, В и Г) и 16 сигналов всего, будет достаточно.

С регистрами и шинами разобрались. Далее у нас Сама АЛУ. Что она должна делать? Естественно Сложение, Вычитание, Инкремент, Декремент, Инвертирование, Побитовое И, Побитовое ИЛИ. Это обязательные функции. Можно подключить оба входа АЛУ к разным шинам. Тогда он сможет работать с любыми регистрами.

Теперь, УУ. Оно должно принимать команды из памяти команд. Обрабатывать их, и управлять всеми остальными элементами ЦП. Так же оно изредка должно выводить команду на шины, для использования её в качестве данных. Насчёт битности надо говорить именно тут. Так как битность команды влияет на общее их количество. Любые данные с битностью ЦП это «слово», запомните, что это не байты, Байты это 8бит. Вспоминаем что я сделал 2 шины из-за производительности, а значит и в командах я не могу использовать 2 слова, ведь их придётся за 2 такта подгружать. Так же не хочется и заходить с битностью слишком далеко, и поэтому я выбираю 8бит. Итого мы получаем ограничение в 255 команд + пустая команда.

Вроде бы всё. То, что могло прийти в голову, мы уже обдумали. Приступаем к проектированию.

**Список команд**

Как я уже решил, что у нас будет восьми-битные команды. То всего команд может быть 256. С учётом той команды, которая ничего не делает (пусть будет иметь код FF), у нас их остаётся 255.

Вместо того чтобы расписывать уже сейчас каждую команду, с начало подумаем. А что должен уметь Процессор чтобы выполнять всё что угодно? Во-первых, работа с регистрами. ЦП должен уметь читать и записывать в регистры данные. Во-вторых, должен выполнять Арифметико-логические действия. В-третьих, Условия. В зависимости от результатов сравнения двух операндов, должен осуществлять переход к нужной команде. И в-четвёртых, должен уметь переходить к определённой команде, без каких-либо условий. Получилось 4 типа команд, которых в достаточном объёме хватит даже для программы извлечения корня.

Теперь подробнее о первом Типе, о командах контроля регистров. Это понятие можно слегка изменить. «Команды миграции данных» звучит лучше, да и по смыслу подходит. Стоп. А как мы будем произвольное значение в регистр записывать? Придётся сделать команды записи в регистр числа, а число это в памяти команд находится. Итого на эту операцию уходит 2 такта. Видимо этого не избежать. О! Мы и про память данных забыли (RAM). Как с ней работать? Её тоже реализуем в этом типе команд. И тут как раз 2 шины нам идут на руку. Мы можем брать адрес из 1 регистра, а считывать или записывать значение из другого регистра. А теперь посчитаем сколько всего команд нужно для всего этого. Перенос информации из одного регистра в другой 16 команд. Чтение и запись в RAM это 16\*2 команд. И запись своего числа, это 4 команды. Всего 52. Это более чем приемлемо.

Второй тип, «Арифметико-логические операции». Тут обязательно есть 7 операций и 4 регистра. Совокупности регистров, то есть какой регистр — это операнд А, а какой операнд Б, всего 16. Умножим это на количество операций и получаем 112. Это слишком много! В голову приходит фиксация АЛУ за Определёнными регистрами. Зафиксирую его за регистрами Ак и А. итого всего 7 команд нужно на управление АЛУ. Но мы ещё не учли, как нам работать непосредственно с регистром Ак. Оно тоже должно принимать и отдавать своё значение, причём работать оно должно с любым регистром. Регистров 4, а операций 2 (принять, отдать значение). И в общем итоге на АЛУ мы используем 8+7+ ещё несколько команд.

Третий тип, «условные операторы». Во-первых, тут сравниваются 2 числа из любых регистров, а минимальное количество сравнений равно трём (>, <, =). 16\*3 команд уже есть. Это очень много. Поэтому надо оптимизировать. Заметьте, что А=Б тоже самое что и Б=А. Применяем сокращение на основе этого факта и получаем 6\*3 команд. Это сокращение сильно усложнит УУ, но зато, теперь есть место для добавления чего-нибудь ещё. А конкретно будет добавлены условия истинности, и работать они будут через булевы переменные. Да и сами переменные должны будут контролироваться. Получается сравнение переменной с единицей, её установка в 0, и в 1. Таких переменных должно быть 8, почему бы и нет. Итого в УУ добавляется ещё 6\*3 + 8\*3 = 42 команд.

Четвёртый тип, наверное, будет состоять из 3х видов команд. «перейти на ячейку, указанную в программе» и «перейти на ячейку, указанную в регистре» и «перейти на нулевую ячейку». Итого команд 1+4+1. Этого вроде бы более-менее хватит.

**Подготовка**

Запускаем программу и приступаем к проектированию! Тут же возникает вопрос. Что с начала делать? И ответ есть! Делай сумматор…

Сумматор — это отличная тренировка для человека, никогда не проектировавшего логические схемы.

Наша подготовка будет иметь для обучения множество Комбинационных элементов:

* Half adder
* Full adder 1bit
* Full adder 4bit
* Fast adder 4bit
* Fast Adder 8bit
* 1-2 decoder
* 2-4 decoder
* 4-16 decoder
* and 8bit
* or 8bit
* not 8bit
* Lock 1bit
* Lock 8bit
* D-trigger
* T-trigger
* Or-trigger

Это все начальные элементы. Они понадобятся нам далее. Без этих базовых схем проектирование новичку очень тяжело будет даваться. Помимо самого наличия схем и возможности их использования мы сможем разобрать их логику и принципы действия. А остальные элементы, которые понадобятся нам далее, мы сделаем по ходу проектирования.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | Fin | d | Fout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FullAdder1bit.png |
| Полный Сумматор |

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FullAdder4bit.png |
| Четырёх битный сумматор |

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HalfAdder.png |
| Полусумматор |

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | d | f |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Полусумматор (Half adder)  
Он умеет складывать 2 однобитных числа  
По его таблице истинности видно как оно должно работать.  
То есть оно выводит на d сигнал только тогда, когда только на 1 входе есть сигнал. А на f появляется сигнал тогда, когда на обоих входах есть сигнал. Отсюда можно вывести формулы.  
Full adder 1bit)  
Он является продолжением полусумматора, но добавляется ещё 1 контакт, который позволяет учитывать перенос при сложении.  
Тут уже таблица больше, из-за большего числа совокупностей сигналов. «HA» это полусумматор (Half Adder).  
  
  
Четырёх битный Сумматор (Full adder 4bit)  
Это последовательность однобитных Сумматоров, соединённых через сигнал переноса. У этого элемента большая таблица истинности, поэтому тут будет только схема.

Быстрый Четырёх битный Сумматор (Fast Adder 4bit)

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FastAdder4bit.png |
| Быстрый Четырёх битный Сумматор |

Это ещё более сложное устройство. Причина его создания заключается в медлительности обычного сумматора. Там сигнал переноса должен был пройти через все разряды, но здесь такого сигнала нет. Результат оценивается из совокупности входящих сигналов, при той же таблице истинности.  
  
  
  
  
  
  
  
  
Быстрый Восьми битный Сумматор (Fast Adder 8bit)

|  |
| --- |
|  |
| Декодер 1 в 2 |

Представляет собой два Fast Adder 4bit, соединённых через флаг переноса.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1-2decoder.png |
| Декодер 1 в 2 |

Декодер 1 в 2 (1-2 decoder)   
Получает на вход однобитное двоичное число и активирует определённый выходной сигнал, соответствующей числу на входе.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\2-4decoder.png |
| Декодер 2 в 4 |

Декодер 2 в 4 (2-4 decoder)  
Получает на вход двух битное двоичное число и активирует определённый выходной сигнал, соответствующей числу на входе.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\4-16decoder.png |
| Декодер 4 в 16 |

Декодер 4 в 16 (4-16 decoder)  
Получает на вход четырёх битное двоичное число и активирует определённый выходной сигнал, соответствующей числу на входе.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\and8bit.png |
| Восьми битное И |

Восьми битное И (And 8bit)  
Проводит одинаковые разряды восьми битного числа через Логический Элемент И.  
Обратите внимание на чёрные элементы. Это разветвители. Через них делается «многожильные провода» или же «многобитные провода». В данном случае там восьми битный провод разделяется на однобитные и наоборот.

Восьми битное ИЛИ (Or 8bit)

|  |
| --- |
|  |
| Восьми битное ИЛИ |

Проводит одинаковые разряды 8 битного числа через Логический Элемент ИЛИ.

Восьми битное НЕ (Not 8bit)

|  |
| --- |
|  |
| Восьми битное НЕ |

Проводит одинаковые разряды 8 битного числа через Логический Элемент НЕ.

Блокировка сигнала (Lock 1bit) (1 variant)

|  |
| --- |
|  |
| Однобитная битная блокировка сигнала |

Данная схема блокирует сигнал **d** при наличии сигнала **x**.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Lock8bit2v.png |
| Восьми битная блокировка сигнала |

Восьми битная блокировка сигнала (Lock 8bit) (2 variant)   
Разрешает сигналы **a** при наличии сигнала **x**.

Асинхронный Триггер (D-trigger)

|  |
| --- |
|  |
| Асинхронный D Триггер |

Это устройство способное запоминать своё состояние.  
  
  
  
  
  
Асинхронный Счётчик (T-trigger)

|  |
| --- |
|  |
| Декодер 1 в 2 |

Это триггер способный, при наличии особого сигнала, менять своё состояние на противоположное. Заметьте, что самим триггером является разновидность D триггера. (обведённое). Всё остальное нужно для создания правильных управляющих сигналов.  
  
Асинхронный ИЛИ триггер (Or-trigger)

|  |
| --- |
|  |
| Декодер 1 в 2 |

Моя собственная разработка. Делает то же самое что и D-trigger, но основан на элементе ИЛИ.

**Реализация**

“Запускаем инструментарий и мозг! Мы погружаемся в Науку!”

Фазылзянов Р.Д.

Первым делом начинаем проектирование АЛУ. Оно у нас должно быть модульным и не обязано поддерживать несколько шин, так как работает только с 1 регистром и Ак за такт, и этого хватит для вычисления. (я же решил, что вычисления только с регистром А и Ак проводятся). Поэтому, на этом этапе создадим шинам названия: «Шина 1» и «Шина 2». Банально, но пусть. АЛУ подключим к второй шине. Так как первую я лучше оставлю под другие задачи.

АЛУ должна:

* Иметь регистр Ак.
* Сравнивать числа. Не используя АЛУ. Причина этому эффективность. Если мы будем создавать условия, то необходимо будет подгрузить данные с Памяти в регистр и далее в Ак. Проще убрать из списка Ак, к тому же, я, как писал ранее, предполагаю отдельную группу команд для условий.
* Производить Сложение
* Производить Вычитание
* Производить Инкремент
* Производить Декремент
* Замену операндов местами.

Так как АЛУ модульная то нужно сделать порядок элементов, ведь мы можем активировать эти модули отдельно, составляя совокупность операций над числом, и игнорируем на этом этапе Устройство сравнения чисел.

Тут есть 2 варианта. В первом, число проходит арифметику, потом логику. Во втором наоборот. И сразу же обращаю ваше внимание на второй вариант. На арифметику приходят 2 числа, и какое из чисел должно было пройти через логику? Сразу 2 или одно, и если одно, то какое именно? Это не только ставит нас перед выбором, но и может усложнить схему управления АЛУ. Выбираем первый вариант. Там нет таких вопросов загвоздок. Есть 1 число на выходе арифметики, оно и проходит через логику.

В арифметической части как первый операнд будет использоваться регистр Ак, а как второй, регистр А. Потом они проходят через устройство смены местами. Далее первый операнд проходит через устройство инвертирования, попадает в Сумматор вместе с неизменённым вторым операндом, и на выходе, первый операнд опять проходит через инвертор.

В логической части будет происходить лишь одна логическая операция. Так мы немного сократим набор команд. Управляться эта часть будет через 2 контакта, или же через двухбитное число. Так мы можем реализовать 4 логические операции, но у нас на данном этапе их всего 3. Вспоминается мне операция следования. Её и добавим.

В добавок нужно учесть тот факт, что я сторонюсь особенности модульной схемы. Возможность обработать числа несколькими операциями за такт заставит увеличить количество команд. Поэтому данное преимущество не используется.

Теперь АЛУ выглядит примерно так:

|  |
| --- |
|  |
| Схема АЛУ использующейся в ЭВМ проекта. С определением значения контактов. |

Многие комбинационные элементы вам не знакомы так как мы их ещё не спроектировали, но этим мы сейчас и займёмся.

Первое это Register. Этот элемент соответствует тому о чём я вам говорил. 4 управляющих сигнала и 5 информационных.

* **get1** –Выводит записанное число на восьми-битный выход **1out**
* **get2** –Выводит записанное число на восьми-битный выход **2out**
* **set1** – Записывает число с восьми-битного входа **1in**
* **set2** – Записывает число с восьми-битного входа **2in**
* **AD** – Всегда выводит записанное число.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Register.png |
| Схема Register использующейся в ЭВМ проекта. |

Первое на что нужно обратить ваше внимание, это обозначенная область на рисунке. Byte представляет собой D триггеры для каждого из 8 разрядов. К тому же они подсоединены последовательно, но последний имеет инвертированный сигнал записи. Такая последовательность D триггеров позволяет записывать данные только в определённый момент. Исключительно при спаде сигнала записи. Так нужно только из-за того, что сам по себе D записывает сигнал на протяжении всего сигнала записи. Так же в схеме много элементов Lock. Это восьми-битные схемы блокировки сигнала. Мы их разбирали. А ещё сверху имеется вход, который всегда выдаёт число 0. Это условность программы, ведь если к входу ничего не подключить, то провод будет передавать значение неподключенного провода, а в таком состоянии Логические элементы ведут себя ненормально. Поэтому я перевожу входы, которые не должны быть обязательно подключены, через оператор ИЛИ с нулём на одном из входов.

|  |  |
| --- | --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Swap Module.png | C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ControlConductor.png |
| Схема SM использующейся в ЭВМ проекта. | Схема СoCo использующейся в ЭВМ проекта. |

Следом идёт Swap Module. Который при наличии сигнала на входе **isS**, меняет местами восьми-битные сигналы **A** и **B**. И имеет в себе Кондукторы (Control Conductor), которые выводят сигнал **A** изначально на выход **q1**, а при наличии сигнала **i1b**, выводит сигнал **A** на выход **q2**. Все входы и выходы тут восьми-битные.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Inverter.png |
| Схема Not использующейся в ЭВМ проекта. |

Not это элемент способный как инвертировать, так и не инвертировать входящий восьми-битный сигнал. Если на входе **NIin** есть сигнал то, инверсия не происходит. Основана логика работы этого элемента на Иск. ИЛИ-НЕ.  
Сделана она через таблицу истинности.

|  |
| --- |
|  |
| Схема Not использующейся в ЭВМ проекта. |

Logic Module. Получает на вход 2 числа, и выводит результат той логической операции, чей код был указан на входах **f**.

Для определения операции стоит декодер, и выводы от него подключены к элементам Lock. Далее идёт разводка из элементов что мы уже знаем. Первое это НЕ **A**. Второе это И между **A** и **B**. Третье, ИЛИ между **A** и **B**. Четвёртое, между **A** и НЕ **B**.

Всё что связанно с АЛУ мы закончили. Поставим теперь регистры. Тут ничего сложного нет. Они все подключены примерно одинаково, но именно сейчас пришло время рассказать тайну. Тайну того, посему я везде вместо объединения проводников, проводил их через ИЛИ. Тут проблема в реализации самих логических элементов. Если просто соединить проводники, то ток в них может подняться в 2 раза. (учим физику если не знаем почему так). Именно поэтому существует Логическая реализация соединения проводников. И если программа в которой вы рисуете схемы, не позволяет вам объединять проводники, то она адекватна. Именно с этой проблемой реализации шины мы сталкиваемся. Нам надо каждый регистр будет подключить через элементы ИЛИ в 1 провод.

|  |
| --- |
|  |
| Строение Шин использующейся в ЭВМ проекта. |

Там мы видим элементы с лампочками. (лампочки для отладки). Внутри них 9 элементов Or 8bit. Каждый вывод на шину должен выходить прямо на этот элемент.

|  |
| --- |
| C:\Users\Ruslan145\AppData\Local\Microsoft\Windows\INetCache\Content.Word\34 Память.png |
| Строение Памяти использующейся в ЭВМ проекта. |

Мы кстати затронули элемент RAM, который к шине подключается точно так же, как и регистры. Но в отличие от регистров, RAM работает сразу с двумя Шинами. Он использует любую из двух шин для передачи или приёма данных, и оставшуюся для приёма адреса ячейки с которой необходимо работать. Но в этом элементе используется встроенная в мою программу разработки схема. Я просто добавил ей расширенное управление.

Далее можно приступить к памяти команд. Адрес текущей команды создаёт счётчик. Оно должно назначаться на любое значение и естественно производить счёт тактовых сигналов. Каждый такт – новый адрес. Этот подход лишает нас возможности потратить 2 и более тактов на 1 команду, но уменьшит количество команд и, следовательно, упростит УУ.

ПЗУ это встроенный модуль в мой редактор, который хранит данные и не позволяет их менять. CoD (Counting Device), тот самый элемент, который считает, и которому можно установить любое число.

|  |  |
| --- | --- |
|  |  |
| Строение Counting Device использующейся в ЭВМ проекта. | Строение T-trigger использующейся в ЭВМ проекта. |

CoD имеет в себе не самый обыкновенный T триггер. Этот триггер сложнее того что я показывал вам ранее. Разница в управлении и приоритете входных сигналов.

**УУ**

Устройство Управления. Самое сложное место любого процессора. Принимает команды и контролирует всё.

УУ у нас будет реализовано в виде Комбинационной схемы, принимающей на вход тактовый сигнал, команду из памяти, некоторые данные от некоторых компонентов ЦП.

|  |
| --- |
|  |
| Строение Разветвителя команд использующейся в ЭВМ проекта. |

В самом УУ нам нужно реализовать более простые сигналы, описывающую входящую команду, да так, чтобы можно было сразу определить тип команды. Для этого можно сразу создать стандарт команд для УУ. Всего у нас 8 разрядов (восьми-битное число), и мы можем разделить его по парам. 00`00`00`00. В первую пару (старшую пару разрядов) мы записываем номер группы команд. Как раз 2 разряда могут передать 4 числа, ровно столько, сколько групп команд. Более младшая пара может указывать на конкретную команду, а две оставшиеся младшие группы могут указывать на регистры, с которыми команда должна работать. И получается, что наше восьми-битное число проходит через 4 дешифратора 2 на 4.

Теперь вопрос по поводу реализации пропуска команд, которое понадобится нам в получении данных из команды, и в условных операторах. Этот элемент самый сложный в УУ. Он завязан на счётчиках и модулях блокировки управляющего сигнала. И тут нам понадобится второй тактовый сигнал.

Загрузку числа из команды в регистр я представляю как последовательность: приходит команда; получаем управляющий сигнал на такте получения управляющих сигналов; в управляющем сигнале активен сигнал «isSave»; блокируем и запоминаем управляющие сигналы; на такте выполнения, управляющие сигналы не передаются; все сигналы тактирования выключаются; опять включается такт получения управляющих сигналов; переход к новой команде; так как управляющие сигналы были сохранены, блокируем команду в самом УУ и выводим её на шину 1; на такте выполнения, разблокируем и выводим управляющие сигналы. Почти это же происходит при отрицательной работе условия. А конкретно различия в том, что там не запоминаются управляющие сигналы и пропускается 2 периода тактов.

|  |
| --- |
|  |
| Строение Устройства Управления использующейся в ЭВМ проекта. |

На рисунке выше, видно готовую компоновку УУ. Там уже всё разведено и готово к работе. Но в самих комбинационных схемах Типов команд нет пока ещё ничего. Да и разберём мы это слегка позже. А сейчас обращаю внимание на левый выделенный блок на этом же рисунке. Это устройство счёта команд и контроля состояний. Оно командует нижним блоком и, сверху, блокировкой команд. Вы можете сказать, что там очень много Or триггеров, но это следствие простого выбора. Мне нужен был триггер, который при подаче питания, запускался бы в нужном мне состоянии. Первый кто из D и Or триггеров попался мне под руку для переделки, был именно Or. И естественно его схема управления стала в разы сложнее.

|  |
| --- |
|  |
| Строение Or-trigger использующейся в ЭВМ проекта. |

Нижний блок на рисунке с УУ управляется левым, как я уже и говорил. Его задача, это приём на входы 1, 2, 3 и 4 четырёхразрядных управляющих сигналов с почти каждой группы команд, сохранение и/или блокировка сигналов.

|  |
| --- |
|  |
| Строение Control Gateway использующейся в ЭВМ проекта. |

**Первый тип команд**

Приступим к проектированию «Устройства первого типа команд». Оно должно управлять регистрами А, Б, В, Г, управлять RAM, сохранять управляющие сигналы с перенаправлением команды на шину. Для достижения этой цели обратимся к параграфу «СПИСОК КОМАНД», где мы разобрали все типы команд разом. Теперь настало время для реализации конкретно этого, первого, типа.

На рисунке 38 видно какие данные передаются в Устройства типов команд. А именно восьми-битное число, делящееся на 4 группы по 2 проводника. Эти группы проходят через дешифратор 2 в 4. Иными словами если по группе передаётся число 2, то и на выходе дешифратора будет активен именно второй выход.

|  |  |
| --- | --- |
| 00101110 | Команда |
| 00 10 11 01 | Группы |
| 0001 0100 1000 0010 | Выход дешифраторов |
| T pT d1 d2 | Названия групп |
| Порядок преобразования команды по пути в устройство типов команд. | |

Разобравшись с входящими данными, нам нужно определить, является ли наше Устройство типа команд, нужным для составления управляющих сигналов при данной команде. Так как мы обозначаем тип нашей команды в двух старших разрядах (группа T), то это делается благодаря простой проверке. Пропускаем все остальные сигналы через Lock схемы, если на входе активирован нужный сигнал. Теперь мы можем как угодно работать с группами pT, d1, d2. Такой механизм мы будем использовать во всех устройствах типов команд. (рис. 41)

Далее требуется упростить работу с регистрами. Это моё личное желание, которое я считаю верным. Заключаться упрощение будет в преобразовании сигналов «Get1», «Get2», «Set1» и «Set2» в сигналы «Enable», «Get», «Set» и «is2». Первый сигнал, «Enable», подразумевает собой разрешение на управление регистром. Второй и третий сигналы, «Get» и «Set», обозначают разрешение на выдачу на шину или запись с шины какого-либо значения. И четвёртый сигнал, «is2», переключает регистр на другую шину. Данная реализация не позволяет регистру одновременно работать с двумя шинами, но управление стало легче за счёт добавления «Enable», и количество сигналов сохранилось за счёт «is2». Это всё будет происходить в новой комбинационной схеме, TRD (Type to Register Decoder). (рис 42)

К слову, в TRD используется схема, не описанная ранее. SiCo (Signal Conductor). Он, в зависимости от состояния входа «is1cable», перенаправляет сигнал с входа «x» на выход «q1» или «q2». (рис 43)

Теперь, прибегнем к логическим функциям, таблицам действий, и анализу. Для каждого выходного контакта требуется логическая функция, без неё проектировать подобные схемы чрезвычайно тяжело. (по сути, логические схемы будут описывать уже готовые управляющие сигналы).

Для составления логических схем на понадобится создать список всех команд определённого типа, вписать необходимые управляющие сигналы, которые должны быть активированы при определённой команде, сгруппировать команды по управляющим сигналам, которые они вызывают, и, наконец, составить и далее сократить логическую функцию для каждого управляющего сигнала.

Итоговая схема выходит довольно простой, но если вы не поняли, то как я пришёл к данной разводке компонентов, то вам необходимо просто посмотреть, на сигналы pT и на то, какие они активируют регистры. Или в крайнем случае, попрактикуйте создание своих дешифраторов. Для продолжения вам будет необходимо знать принцип и логику работы таких схем, ведь, порой, исключительно на дешифраторах бывают основаны некоторые схемы, которые вовсе не работают по своей таблице истинности как дешифратор. (рис. 44)

Так же прошу вас заметить, что в Устройстве первого типа команд с двух TRD соединяются одноимённые выходы. Таким образом устраняется проблема ограничения работы регистра с шинами, вызванная самим TRD. Вернул я эту возможность из-за команд, что требовали для своей реализации работу регистра с несколькими шинами одновременно. Можно было бы всё переделать, да и не сложно это будет сделать, но схема работает и менять я её не буду. Любая победа или оплошность, это опыт.

**Второй тип команд**

Второй тип команд отвечает за работу АЛУ и регистра Ак. В данном случае при реализации этой схемы, были созданы промежуточные шины (смотрите на рисунке 45 выделенные области), которые хранили некоторую логику. Например, у нас есть две логические функции: (A+B)\*C и C\*D\*(B+A). Сейчас для реализации этих функций нам нужно 4 Логических элемента. Выведем результат A+B как отдельный сигнал, и назовём его E. Тогда мы сможем преобразовать функции в следующие: E\*C, C\*D\*E и E=A+B. И теперь нам необходимо уже 3 логических элемента для реализации этого же функционала. Были применены простейшие сокращения. Скопление таких промежуточных результатов в одном месте можно назвать промежуточной шиной, но если такой промежуточный результат поступает только в 1 вход (используется в одном месте), то его нельзя считать промежуточным. (рис. 45)

В отличие от Устройства первой группы команд, в этом устройстве TRD не мешали, а наоборот помогли. Всё же созданы они были не зря, и заложенная в них функция пригодна к использованию.

**Третий тип команд**

Самый сложный в реализации тип команд. Не буду тянуть резину, и сразу говорю вам посмотреть на рисунок 46. Здесь предыдущими вариантами обойтись не вышло. В отличии от других типов команд, здесь не удалось все команды грамотно разделить среди групп T, pT, d1 и d2. Их пришлось заново объединять в шести-битное число (выделенное слева). И расписывать каждую из 42 команд в этом шести-битном числе.

После числа, теперь обозначающего номер команды, стоит два дешифратора 5 в 32, и от них разведены пересекающиеся матрицы. Одна такая матрица обозначает команды с одинаковым принципом работы, но с слегка разной логикой. Даже есть три матрицы, переплетённые между собой (выделенное снизу).

Особенно обращаю ваше внимание на «Условную матрицу» (выделенное сверху). Принимая команду, она производит проверку с булевыми переменными. Но вот только на моменте итоговой проверки этого проекта я заметил, что матрица чуть ниже с элементом ИЛИ, не требуется. Как причину этой оплошности могу назвать избыток проводов в схеме.

**Четвёртый тип команд**

Пожалуй, самый лёгкий тип команд, в основном из-за малого количества команд и управляющих сигналов. Напрямую влияет на работу счётчика команд. И не имея промежуточных посредников формирует сигнал «setAddres». (рис. 47)

Так же, как и Устройство первого типа команд, имеет команды содержащие данные, и выдаёт те же сигналы «isSave» и «comToBus».

**ИТОГ**

ЭВМ готова. Связав все компоненты вместе, мы получаем абсолютно рабочую Электронно-Вычислительную Машину! Правда, без органов управления, но они не обязательны. У этой машины высокий потенциал по модернизации, переработке и исправлениям. Назову её «ЭВМ им. Фазылзянова Р.Д.» (рис. 48)

Человек, разобравшийся в данной науке, сможет проектировать подобные и не только вещи как для своей нужды, так и для коммерции, главное, чтобы энтузиазма, знаний хватило. Подобным образом становились те, кто эту отрасль закрыл. Поэтому на энтузиазм сейчас вся надежда.

Предполагаю, что в данном проекте высказал всё что необходимо для старта и вхождения в эту отрасль на достаточно высоком уровне. Но к сожалению, человек впервые слышащий об этой отрасли не сможет прочесть мой проект и вникнуть в суть. Сделано это из-за доступности той информации, которая необходима читателю этого проекта, ведь если человеку стало интересно, и он хочет повысить свой уровень знаний, то мой проект ему поможет как текстом, так и примерами и собственно схемами. К слову, проектом я занимался, суммарно, 6 месяцев.

Два продукта моего проекта успешно считаю законченными: Книга о проектировании собственной архитектуры ЭВМ и собственно сама схема и архитектура ЭВМ. Они будут относится как материалы этого проекта и будут находиться в приложении ниже.

Благодарю за прочтение.