Оглавление

ВВЕДЕНИЕ

В рамках выполнения лабораторных работ и курсового проектирования необходимо разработать программную модель простейшей вычислительной машины Simple Computer. Архитектура Simple Computer представлена ниже.

Для управления моделью (определения начальных состояний узлов Simple Computer, запуска программ на выполнения, отражения хода выполнения программ) требуется создать консоль (см. рисунок 1). Необходимо реализовать трансляторы с языков Simple Assembler и Simple Basic[1] для программирования Simple Computer.

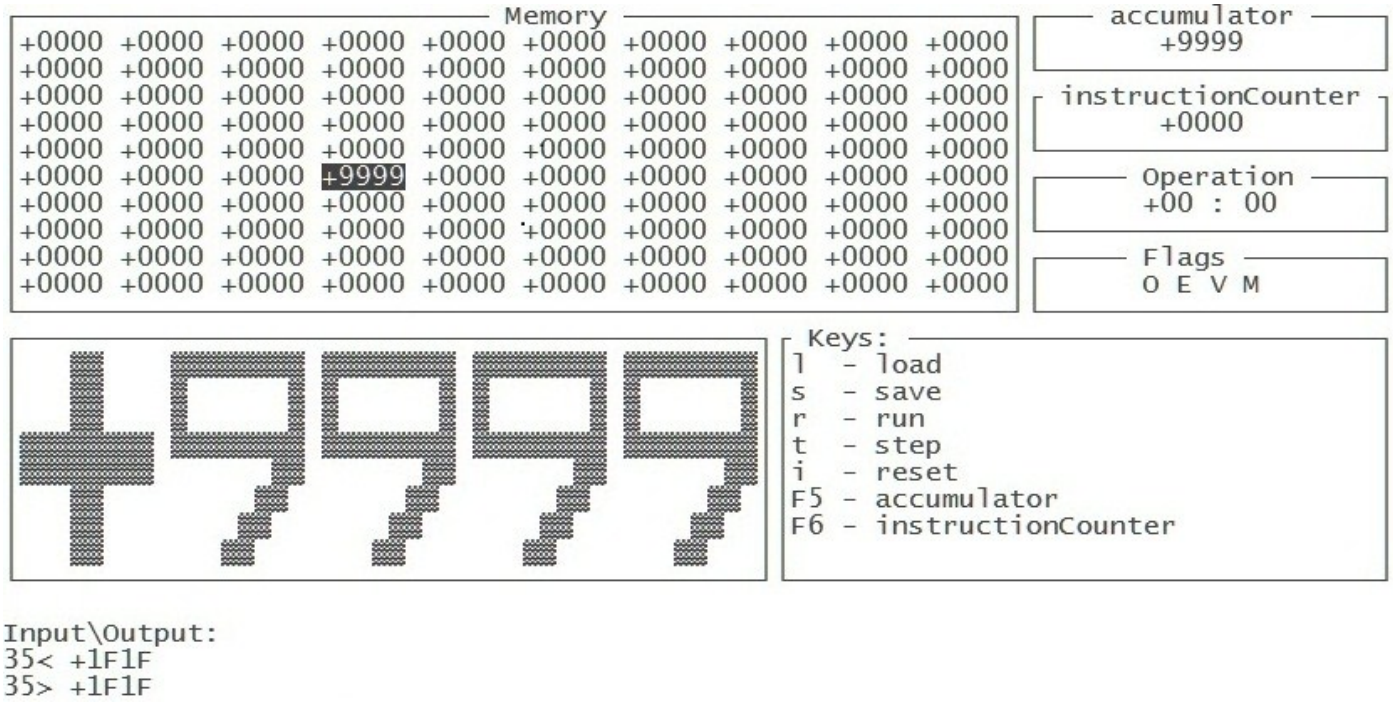

Рисунок 1 – интерфейс консоли управления моделью Simple Computer

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ И КУРСОВОГО ПРОЕКТИРОВАНИЯ

Нед	Лабораторные работы	Курсовая работа
1	Введение в курс. Знакомство со средой разработки в GNU/Linux. Постановка задачи на лабораторные работы и курсовое проектирование. Лабораторная работа 1. Организация современных персональных компьютеров	
2 3	Лабораторная работа 2. Разработка библиотеки mySimpleComputer. Оперативная память, регистр флагов, декодирование операций.	Выполнение курсового проектирования
4 5	Лабораторная работа 3. Консоль управления моделью Simple Computer. Текстовая часть.	
6 7	Лабораторная работа 4. Консоль управления моделью Simple Computer. Псевдографика. «Большие символы».	
8 9	Лабораторная работа 5. Консоль управления моделью Simple Computer. Клавиатура. Обработка нажатия клавиш. Неканонический режим работы терминала	
10 11	Лабораторная работа 6. Подсистема прерываний ЭВМ. Сигналы и их обработка.	
12	Защита рефератов	
13 14 15 16	Лабораторная работа 7. Устройство хранения данных на жестких магнитных дисках.	Защита курсовой работы
17	Сдача долгов	
18	Сдача долгов	

[1] Транслятор с языка Simple Basic разрабатывают студенты, претендующие на отличную оценку по курсовой работы и «автомат» на экзамене.

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ SIMPLE COMPUTER

Архитектура Simple Computer представлена на рисунке 2 и включает следующие функциональные блоки:

- оперативную память;
- внешние устройства;
- центральный процессор.

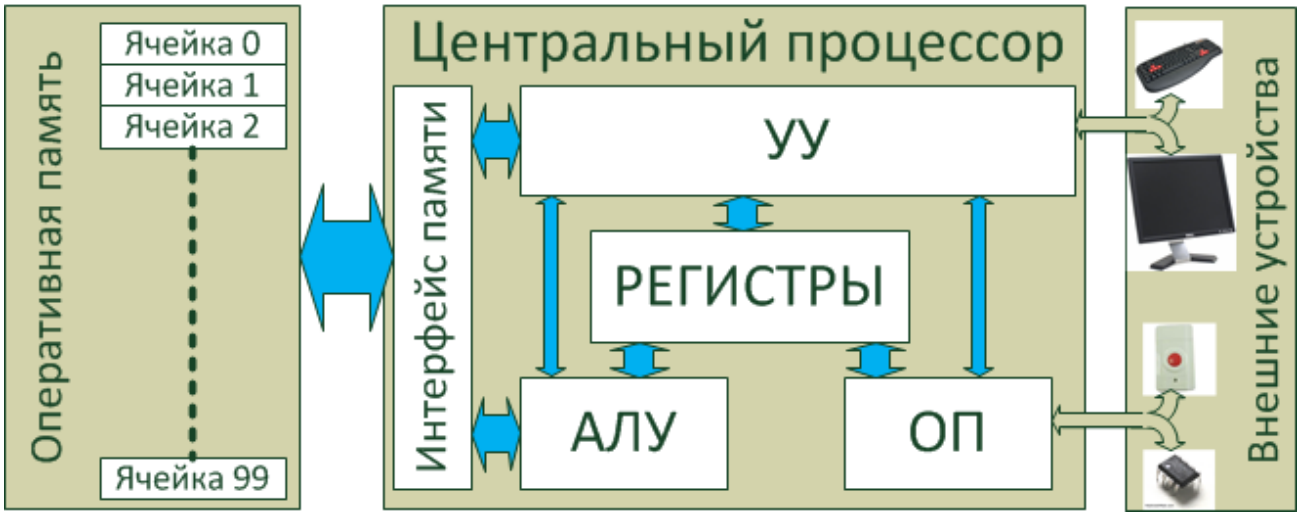

Рисунок 2 – Архитектура вычислительной машины Simple Computer

Оперативная память

Оперативная память – это часть Simple Computer, где хранятся программа и данные. Память состоит из ячеек (массив), каждая из которых хранит 15 двоичных разрядов. Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является её адресом и задается 7-миразрядным числом. Предполагаем, что Simple Computer оборудован памятью из 100 ячеек (с адресами от 0 до 99_{10}).

Внешние устройства

Внешние устройства включают: клавиатуру и монитор, используемые для взаимодействия с пользователем, системный таймер, задающий такты работы Simple Computer и кнопку «Reset», позволяющую сбросить Simple Computer в исходное состояние.

Центральный процессор

Выполнение программ осуществляется центральным процессором Simple Computer. Процессор состоит из следующих функциональных блоков:

- регистры (аккумулятор, счетчик команд, регистр флагов);
- арифметико-логическое устройство (АЛУ);
- управляющее устройство (УУ);
- обработчик прерываний от внешних устройств (ОП);
- интерфейс доступа к оперативной памяти.

Регистры являются внутренней памятью процессора. Центральный процессор Simple Computer имеет: аккумулятор, используемый для временного хранения данных и результатов операций, счетчик команд, указывающий на адрес ячейки памяти, в которой хранится текущая выполняемая команда и регистр флагов, сигнализирующий об определённых событиях. Аккумулятор имеет разрядность 15 бит, счетчика команд – 7 бит. Регистр флагов содержит 5 разрядов: переполнение при выполнении операции, ошибка деления на 0, ошибка выхода за границы памяти, игнорирование тактовых импульсов, указана неверная команда.

Арифметико-логическое устройство (англ. arithmetic and logic unit, ALU) — блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде команды или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в акку-

муляторе или может помещаться в оперативную память. В ходе выполнения операций АЛУ устанавливает значения флагов «деление на 0» и «переполнение».

Управляющее устройство (англ. control unit, CU) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. В его функции входит: чтение текущей команды из памяти, её декодирование, передача номера команды и операнда в АЛУ, определение следующей выполняемой команды и реализации взаимодействий с клавиатурой и монитором. Выбор очередной команды из оперативной памяти производится по сигналу от системного таймера. Если установлен флаг «игнорирование тактовых импульсов», то эти сигналы устройством управления игнорируются. В ходе выполнения операций устройство управления устанавливает значения флагов «указана неверная команда» и «игнорирование тактовых импульсов».

Обработчик прерываний реагирует на сигналы от системного таймера и кнопки «Reset». При поступлении сигнала от кнопки «Reset» состояние процессора сбрасывается в начальное (значения всех регистров обнуляется и устанавливается флаг «игнорирование сигналов от таймера»). При поступлении сигнала от системного таймера, работать начинает устройство управления.

Система команд Simple Computer

Получив текущую команду из оперативной памяти, устройство управления декодирует её с целью определить номер функции, которую надо выполнить и операнд. Формат команды следующий (см. рисунок 3): старший разряд содержит признак команды (0 – команда), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд. Коды операций, их назначение и обозначение в Simple Assembler и приведены в таблице 1.

Рисунок 3 – Формат команды центрального процессора Simple Computer

Таблица 1. Команды центрального процессора Simple Computer

Операция		**Значение**
Обозначение	**Код**	
Операции ввода/вывода		
READ	10	Ввод с терминала в указанную ячейку памяти с контролем переполнения
WRITE	11	Вывод на терминал значение указанной ячейки памяти
Операции загрузки/выгрузки в аккумулятор		
LOAD	20	Загрузка в аккумулятор значения из указанного адреса памяти
STORE	21	Выгружает значение из аккумулятора по указанному адресу памяти
Арифметические операции		
ADD	30	Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе)
SUB	31	Вычитает из слова в аккумуляторе слово из указанной ячейки памяти (результат в аккумуляторе)
DIVIDE	32	Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
MUL	33	Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
Операции передачи управления		
JUMP	40	Переход к указанному адресу памяти
JNEG	41	Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число
JZ	42	Переход к указанному адресу памяти, если в аккумуляторе находится ноль
HALT	43	Останов, выполняется при завершении работы программы

Пользовательские функции		
NOT	51	Двоичная инверсия слова в аккумуляторе и занесение результата в указанную ячейку памяти
AND	52	Логическая операция И между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
OR	53	Логическая операция ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
XOR	54	Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
JNS	55	Переход к указанному адресу памяти, если в аккумуляторе находится положительное число
JC	56	Переход к указанному адресу памяти, если при сложении произошло переполнение
JNC	57	Переход к указанному адресу памяти, если при сложении не произошло переполнение
JP	58	Переход к указанному адресу памяти, если результат предыдущей операции четный
JNP	59	Переход к указанному адресу памяти, если результат предыдущей операции нечетный
CHL	60	Логический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
SHR	61	Логический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)
RCL	62	Циклический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
RCR	63	Циклический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)
NEG	64	Получение дополнительного кода содержимого указанной ячейки памяти (результат в аккумуляторе)
ADDC	65	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
SUBC	66	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
LOGLC	67	Логический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
LOGRC	68	Логический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
RCCL	69	Циклический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
RCCR	70	Циклический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов указанное в аккумуляторе (результат в аккумуляторе)
MOVA	71	Перемещение содержимого указанной ячейки памяти в ячейку, адрес которой указан в аккумуляторе
MOVR	72	Перемещение содержимого ячейки памяти, адрес которой содержится в аккумуляторе в указанную ячейку памяти.
MOVCA	73	Перемещение содержимого указанной ячейки памяти в ячейку памяти, адрес которой находится в ячейке памяти, на которую указывает значение аккумулятора
MOVCR	74	Перемещение в указанный участок памяти содержимого участка памяти, адрес которого находится в участке памяти указанном в аккумуляторе
ADDC	75	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)
SUBC	76	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)

Выполнение команд центральным процессором Simple Computer

Команды выполняются последовательно. Адрес ячейки памяти, в которой находится текущая выполняемая команда, задается в регистре «Счетчик команд». Устройство управления запрашивает содержимое указанной ячейки памяти и декодирует его согласно используемому формату команд. Получив код операции, устройство управления определяет, является ли эта операция арифметико-логической. Если да, то выполнение операции передается в АЛУ. В противном случае операция выполняется устройством управления. Процедура выполняется до тех пор, пока флаг «останов» не будет равен 1.

Консоль управления

Интерфейс консоли управления представлен на рисунке 1. Он содержит следующие области:

- “Memory” – содержимое оперативной памяти Simple Computer.
- “Accumulator” – значение, находящееся в аккумуляторе;
- “instructionCounter” – значение регистра «счетчик команд»;
- “Operation” – результат декодирования операции;

- “Flags” – состояние регистра флагов («П» - переполнение при выполнении операции, «0» - ошибка деления на 0, «М» - ошибка выхода за границы памяти, «Т» - игнорирование тактовых импульсов, «Е» - указана неверная команда);
- “Cell” – значение выделенной ячейки памяти в области “Memory” (используется для редактирования);
- “Keys” – подсказка по функциональным клавишам;
- “Input/Otput” – область, используемая Simple Computer в процессе выполнения программы для ввода информации с клавиатуры и вывода её на экран.

Содержимое ячеек памяти и регистров центрального процессора выводится в декодированном виде. При этом, знак «+» соответствует значению 0 в поле «признак команды», следующие две цифры – номер команды и затем операнд в шестнадцатеричной системе счисления.

Пользователь имеет возможность с помощью клавиш управления курсора выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу “F5”, пользователь может задать значение аккумулятору, “F6” – регистру «счетчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может, нажав на клавиши «l», «s» соответственно (после нажатия в поле Input/Output пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорировать такты таймера» в 0) можно с помощью клавиши “r”. В процессе выполнения программы, редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу “t”. Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу “i”.

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1. Организация современных персональных компьютеров

Цель работы

Изучить архитектуру современных персональных компьютеров. Понять назначение основных функциональных блоков, интерфейсов их взаимодействия и схему работы центрального процессора и оперативной памяти.

Задание на лабораторную работу

1. Прочитайте главы 1, 2 и 3 практикума по курсу «Организация ЭВМ и систем».
2. Напишите реферат на одну из приведённых ниже тем согласно Вашему варианту задания. Вариант выбирается исходя из двух последних цифр Вашего учетного имени по модулю количества тем.
3. Изучите устройство материнской платы персонального компьютера на примере любой современной модели.

Темы рефератов

1. Интерфейс конфигурации и управления питанием (ACPI).
2. Процессоры семейства Intel. История развития.
3. Система команд процессоров семейства Intel.
4. Микроархитектура Intel Net Burst.
5. Микроархитектура Intel Core.
6. Микроархитектура Intel Atom.
7. Микроархитектура Intel Nehalem.
8. Архитектура Intel x2APIC.
9. Технология Hyper Threading.
10. Архитектура IA64 (Itanium 2, VLIW, Intel EPIC)
11. Сравнение технологий IA32 и EM64T (Intel 64).
12. Аппаратная поддержка виртуализации в процессорах Intel (Intel VT)
13. Аппаратная поддержка виртуализации в процессорах AMD (AMD-V)
14. Расширение системы команд Streaming SIMD Extension (SSE, SSE2, SSE3, SSE4).
15. Расширение системы команд MMX, 3DNow.
16. Интерфейс PCI, PCI-Express.
17. Интерфейс USB.
18. Интерфейс Bluetooth.

19. Интерфейсы D-Sub и DVI.
20. Интерфейс IEEE1394 (FireWire).
21. Принципы работы процессорного кэша.
22. Предсказание переходов. Спекулятивное выполнение.
23. Параллелизм уровня команд (Instruction Level Parallelism, ILP). Конвейеризация.
24. Параллелизм уровня потоков (Thread Level Parallelism). Технология
25. Параллелизм уровня заданий. Многоядерные процессоры.
26. Шина Hyper Transport
27. Внешняя память. Дисковые массивы RAID.
28. Оперативная память. Чип SPD.
29. Набор микросхем системной логики. Архитектура. Примеры существующих чипсетов.

Процедура защиты реферата

Защита реферата производится в сроки, указанные в календарном плане и проводится в два этапа:

1. беседа по представленному в реферате материалу;
2. рассказ «устройство узлов персонального компьютера».

На втором этапе студентам предоставляется один из узлов персонального компьютера и требуется рассказать назначение этого устройства и его основные характеристики.

Контрольные вопросы

1. Что такое ЭВМ? Персональный компьютер?
2. Зачем нужна материнская плата?
3. Зачем используется блок питания? Корпус?
4. Что такое набор микросхем системной логики?
5. Что такое форм-фактор?
6. Сколько шин в персональном компьютере? Зачем они нужны? Как определить пропускную способность шины?
7. Виды памяти? Статическая и динамическая память?
8. Что такое интерфейс? Какие интерфейсы используются в ПК?

Лабораторная работа 2. Разработка библиотеки mySimpleComputer. Оперативная память, регистр флагов, декодирование операций.

Цель работы

Изучить принципы работы оперативной памяти. Познакомиться с разрядными операциями языка Си. Разработать библиотеку mySimpleComputer, включающую функции по декодированию команд, управлению регистрами и взаимодействию с оперативной памятью.

Задание на лабораторную работу

1. Прочитайте главу 4 практикума по курсу «Организация ЭВМ и систем». Изучите принципы работы разрядных операций в языке Си: как можно изменить значение указанного разряда целой переменной или получить его значение. Вспомните, как сохранять информацию в файл и считывать её оттуда в бинарном виде.
2. Разработайте функции по взаимодействию с оперативной памятью, управлению регистром флагов и кодированию/декодированию команд:
    a. `int sc_memoryInit ()` – инициализирует оперативную память Simple Computer, задавая всем её ячейкам нулевые значения. В качестве «оперативной памяти» используется массив целых чисел, определенный статически в рамках библиотеки. Размер массива равен 100 элементам.
    b. `int sc_memorySet (int address, int value)` – задает значение указанной ячейки памяти как value. Если адрес выходит за допустимые границы, то устанавливается флаг «выход за границы памяти» и работа функции прекращается с ошибкой;
    c. `int sc_memoryGet (int address, int * value)` – возвращает значение указанной ячейки памяти в value. Если адрес выходит за допустимые границы, то устанавливается флаг «выход за границы памяти» и работа функции прекращается с ошибкой. Значение value в этом случае не изменяется.
    d. `int sc_memorySave (char * filename)` – сохраняет содержимое памяти в файл в бинарном виде (используя функцию write или fwrite);

e. `int sc_memoryLoad (char * filename)` – загружает из указанного файла содержимое оперативной памяти (используя функцию read или fread);
f. `int sc_regInit (void)` – инициализирует регистр флагов нулевым значением;
g. `int sc_regSet (int register, int value)` – устанавливает значение указанного регистра флагов. Для номеров регистров флагов должны использоваться маски, задаваемые макросами (#define). Если указан недопустимый номер регистра или некорректное значение, то функция завершается с ошибкой.
h. `int sc_regGet (int register, int * value)` – возвращает значение указанного флага. Если указан недопустимый номер регистра, то функция завершается с ошибкой.
i. `int sc_commandEncode (int command, int operand, int * value)` – кодирует команду с указанным номером и операндом и помещает результат в value. Если указаны неправильные значения для команды или операнда, то функция завершается с ошибкой. В этом случае значение value не изменяется.
j. `int sc_commandDecode (int value, int * command, int * operand)` – декодирует значение как команду Simple Computer. Если декодирование невозможно, то устанавливается флаг «ошибочная команда» и функция завершается с ошибкой.

3. Оформите разработанные функции как статическую библиотеку. Подготовьте заголовочный файл для неё.

Защита лабораторной работы

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях).

Контрольные вопросы

1. Что такое вентиль? Какие значения он может принимать?
2. Сколько вентилей необходимо, чтобы получить логические функции НЕ, ИЛИ-НЕ, И-НЕ, И, ИЛИ?
3. Что такое таблица истинности? Булева функция? Как они связаны между собой?
4. Как получить алгебраическую булеву функцию из таблицы истинности? И наоборот?
5. Каким образом можно синтезировать логическую схему по таблице истинности? По алгебраической формуле?
6. Что такое система счисления? Чем отличается позиционная система счисления от непозиционной?
7. Как получить качественный эквивалент числа в непозиционной системе счисления? В позиционной?
8. Как перевести числа из двоичной системы счисления в десятичную? Восьмеричную? Шестнадцатеричную? И наоборот?
9. Что такое двоично-десятичное число?
10. Как в ЭВМ представляются отрицательные числа и числа с плавающей запятой?
11. Что такое дополнительный код? Зачем он используется?
12. Как перевести десятичное число с плавающей запятой в двоичное?
13. Какие базовые типы данных используются для хранения переменных в языке СИ?
14. Что такое флаг? Зачем он используется? Каким образом можно манипулировать флагами? Что такое маска?

Лабораторная работа 3. Консоль управления моделью Simple Computer. Текстовая часть.

Цель работы

Изучить принципы работы терминалов ЭВМ в текстовом режиме. Понять, каким образом кодируется текстовая информация и как с помощью неё можно управлять работой терминалов. Разработать библиотеку функций myTerm, включающую базовые функции по управлению текстовым терминалом (очистка экрана, позиционирование курсора, управления цветом). Начать разрабатывать консоль управления Simple Computer (вывести на экран текстовую часть).

Задание на лабораторную работу

1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграфы 5.4 и 5.5. Изучите страницу `man` для команды `infocmp`, базы `terminfo`, функции `ioctl`.
2. Откройте текстовый терминал и запустите оболочку `bash` (оболочка запускается автоматически). Используя команду `infocmp`, определите (и перепишите их себе) escape-последовательности для терминала, выполняющие следующие действия:
   - очистка экрана и перемещение курсора в левый верхний угол (clear_screen);
   - перемещение курсора в заданную позицию экрана (cursor_address);
   - задание цвета последующих выводимых символов (set_a_background);
   - определение цвета фона для последующих выводимых символов (set_a_foreground);
   - скрытие и восстановление курсора (cursor_invisible, cursor_visible).
3. Используя оболочку bash, команду echo –e и скрипт[2], проверьте работу полученных последовательностей. Символ escape задается как \033 или \E. Например – echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
   - очищает экран;
   - выводит в пятой строке, начиная с 10 символа Ваше имя красными буквами на черном фоне;
   - в шестой строке, начиная с 8 символа Вашу группу зеленым цветом на белом фоне;
   - перемещает курсор в 10 строку, 1 символ и возвращает настройки цвета в значения «по умолчанию».
4. Разработать следующие функции:
   - `int mt_clrscr (void)` - производит очистку и перемещение курсора в левый верхний угол экрана;
   - `int mt_gotoXY (int, int)` - перемещает курсор в указанную позицию. Первый параметр номер строки, второй - номер столбца;
   - `int mt_getscreensize (int * rows, int * cols)` - определяет размер экрана терминала (количество строк и столбцов);
   - `int mt_setfgcolor (enum colors)` - устанавливает цвет последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала;
   - `int mt_setbgcolor (enum colors)` - устанавливает цвет фона последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала.

   Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода.
5. Оформите разработанные функции как статическую библиотеку myTerm. Подготовьте заголовочный файл для неё.

Защита лабораторной работы

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях), а также программу, выводящую на экран согласно рисунку 1 содержимое оперативной памяти, регистров и назначение клавиш.

Контрольные вопросы

1. Взаимодействие с устройствами в Linux. Специальные файлы устройств.
2. Функции open, close, read, write.
3. Терминалы. Типы терминалов. Эмуляция терминала. Режимы работы.
4. Управление терминалом. Команды. Низкоуровневое управление.
5. Что такое escape-последовательность?
6. Как определить escape-последовательности для терминала?

[2] Скрипт – это текстовый файл, содержащий команды оболочки. Запускается на выполнение командой bash имя_файла.

Лабораторная работа 4. Консоль управления моделью Simple Computer. Псевдографика. «Большие символы».

Цель работы

Изучить работу текстового терминала с псевдографическими символами. Понять, что такое шрифт и как он используется в терминалах при выводе информации. Разработать библиотеку myBigChars, реализующую функции по работе с псевдографикой и выводу «больших символов» на экран. Доработать консоль управления Simple Computer так, чтобы выводились псевдографические элементы.

Задание на лабораторную работу.

1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграфы 5.2, 5.3, 5.4.2. Изучите страницу `man` для команды `infocmp`, базы `terminfo` (раздел псевдографика).
2. Используя оболочку `bash` и команду `infocmp`, определите escape-последовательности для переключения используемых терминалом кодировочных таблиц (enter_alt_charset_mode и exit_alt_charset_mode) и соответствие символов для вывода псевдографики (acs_chars).
3. Используя оболочку `bash`, команду echo –e и скрипт, проверьте работу полученных последовательностей. Символ escape задается как \033 или \E. Например - echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
   - очищает экран;
   - выводит псевдографическую рамку, начиная с 5 символа 10 строки, размером 8 строк на 8 столбцов;
   - с помощью псевдографического символа «закрашенный прямоугольник» (ACS_CKBOARD) в рамке выводится большой символ, соответствующий последней цифре дня вашего рождения (например, день рождения 13 января 1991 года, выводится цифра 3).
4. Разработать следующие функции:
   - `int bc_printA (char * str)` - выводит строку символов с использованием дополнительной кодировочной таблицы;
   - `int bc_box(int x1, int y1, int x2, int y2)` – выводит на экран псевдографическую рамку, в которой левый верхний угол располагается в строке x1 и столбце y1, а её ширина и высота равна y2 столбцов и x2 строк;
   - `int bc_printbigchar (int [2], int x, int y, enum color, enum color)` - выводит на экран "большой символ" размером восемь строк на восемь столбцов, левый верхний угол которого располагается в строке x и столбце y. Третий и четвёртый параметры определяют цвет и фон выводимых символов. "Символ" выводится исходя из значений массива целых чисел следующим образом. В первой строке выводится 8 младших бит первого числа, во второй следующие 8, в третьей и 4 следующие. В 5 строке выводятся 8 младших бит второго числа и т.д. При этом если значение бита = 0, то выводится символ "пробел", иначе - символ, закрашивающий знакоместо (ACS_CKBOARD);
   - `int bc_setbigcharpos (int * big, int x, int y, int value)` - устанавливает значение знакоместа "большого символа" в строке x и столбце y в значение value;
   - `int bc_getbigcharpos(int * big, int x, int y, int *value)` - возвращает значение позиции в "большом символе" в строке x и столбце y;
   - `int bc_bigcharwrite (int fd, int * big, int count)` - записывает заданное число "больших символов" в файл. Формат записи определяется пользователем;
   - `int bc_bigcharread (int fd, int * big, int need_count, int * count)` считывает из файла заданное количество "больших символов". Третий параметр указывает адрес переменной, в которую помещается количество считанных символов или 0, в случае ошибки.

   Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода.
5. Оформите разработанные функции как статическую библиотеку myBigChars. Подготовьте заголовочный файл для неё.

Защита лабораторной работы

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях). Необходимо доработать программу лабораторной работы 3, выводящую на экран согласно рисунку 1 содержимое оперативной памяти, регистров и назначение клавиш, так, чтобы на экране были нарисованы рамки, и выводилось большими символами содержимое ячейки памяти, на которую указывает регистр “instructionCounter”.

Контрольные вопросы

1. Что такое шрифт? Как он используется при выводе символов на экран?
2. Зачем используется кодировочная таблица символов? Какие таблицы Вы знаете?
3. Почему символы, рисующие рамку в текстовом режиме, называются «псевдографическими»?

Лабораторная работа 5. Консоль управления моделью Simple Computer. Клавиатура. Обработка нажатия клавиш. Неканонический режим работы терминала

Цель работы

Изучить устройство клавиатуры и принципы обработки нажатия клавиш в текстовом терминале. Создать «распознаватель» нажатой клавиши по формируемой последовательности символов. Разработать библиотеку myReadkey. Доработать интерфейс консоли управления Simple Computer так, чтобы можно было изменять значения ячеек памяти и регистров.

Задание на лабораторную работу

1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграф 5.1. Изучите страницу `man` для команд `infocmp` и `read`, базы `terminfo`.
2. Используя оболочку `bash` и команду `read`, определите последовательности, формируемые нажатием на буквенно-цифровые, функциональные клавиши и клавиши управления курсором. Используя команду `infocmp`, убедитесь, что получены правильные последовательности символов, генерируемые функциональными клавишами «F5» и «F6».
3. Разработайте функции:
    - `int rk_readkey (enum keys *)` - анализирующую последовательность символов (возвращаемых функцией `read` при чтении с терминала) и возвращающую первую клавишу, которую нажал пользователь. В качестве параметра в функцию передаётся адрес переменной, в которую возвращается номер нажатой (enum keys – перечисление распознаваемых клавиш);
    - `int rk_mytermsave (void)` - сохраняет текущие параметры терминала;
    - `int rk_mytermrestore (void)` - восстанавливает сохранённые параметры терминала.
    - `int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)` - переключает терминала между режимами. Для неканонического режима используются значения второго и последующего параметров.
4. Оформите разработанные функции как статическую библиотеку myReadkey. Подготовьте заголовочный файл для неё.

Защита лабораторной работы

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях). Необходимо доработать программу лабораторной работы 3, выводящую на экран согласно рисунку 1 консоль управления Simple Computer так, чтобы возможно было задавать значения ячейкам оперативной памяти, регистрам и обрабатывалось нажатие клавиш “s”, “l”.

Контрольные вопросы

1. Режимы работы терминала. Как настроить терминал для работы в неканоническом режиме?
2. Работа с терминалом в Linux. Структура termios.

Лабораторная работа 6. Подсистема прерываний ЭВМ. Сигналы и их обработка.

Цель работы

Изучить принципы работы подсистемы прерываний ЭВМ. Понять, как обрабатываются сигналы в Linux. Реализовать обработчик прерываний в модели Simple Computer. Доработать модель Simple Computer, создав обработчик прерываний от внешних устройств «системный таймер» и «кнопка».

Задание на лабораторную работу

1. Прочитайте главу 6 практикума по курсу «Организация ЭВМ и систем». Изучите страницу `man` для функций `signal`, `setitimer`.
2. Доработайте консоль Simple Computer. Создайте обработчик прерываний от системного таймера так, чтобы при каждом его срабатывании при нулевом значении флага «игнорирование сигналов системного таймера» значение регистра “instructionCounter” увеличивалось на 1, а при поступлении сигнала SIGUSR1 состояние Simple Computer возвращалось в исходное. Обработка нажатых клавиш осуществляется только в случае, если сигналы от таймера не игнорируются.

Защита лабораторной работы

Для защиты лабораторной работы необходимо подготовить программу, реализующие консоль управления Simple Computer и демонстрирующую работу обработчика прерываний.

Контрольные вопросы

1. Что такое прерывание? Что такое сигнал? Чем они отличаются друг от друга? Какую информацию несут в себе прерывание и сигнал?
2. Как происходит обработка сигнала в программах, работающих под управлением ОС Linux?
3. Каким образом настраивается таймер? Как программа «узнаёт» о срабатывании таймера?
4. Каким образом пользовательская программа может узнать об изменении размера окна виртуального терминала?

Лабораторная работа 7. Устройство хранения данных на жестких магнитных дисках.

Цель работы

Изучить устройство накопителей на жестких магнитных дисках. Разработать библиотеку функций по преобразованию геометрий и адресов секторов накопителей на жестких магнитных дисках. Создать программу, рассчитывающую таблицу разделов (MBR).

Задание на лабораторную работу

1. Прочитайте главу 7 практикума по курсу «Организация ЭВМ и систем».
2. Разработайте пользовательские типы (typedef), для хранения адресов секторов и геометрий жестких дисков в форматах:
   - CHS (20 бит). Имя пользовательского типа - tCHS;
   - ECHS или Large (24 бита). Имя пользовательского типа - tLARGE;
   - CHS из стандарта IDE (28 бит). Имя пользовательского типа tIDECHS;
   - LBA (32 бит). Имя пользовательского типа - tLBA.
3. Создайте библиотеку функций по преобразованию геометрий и адресов секторов накопителей на жестких магнитных дисках в разные стандарты:
   - `int g_lba2chs (tLBA, tCHS *).`
   - `int g_lba2large (tLBA, tLARGE *).`
   - `int g_lba2idechs (tLBA, tIDECHS *).`
   - `int g_chs2large (tCHS,  tLARGE *).`
   - `int g_chs2lba (tCHS, tLBA *).`
   - `int g_chs2idechs (tIDECHS, tLBA *).`
   - `int g_large2chs (tLARGE, tCHS *).`
   - `int g_large2idechs (tLARGE, tIDECHS *).`
   - `int g_large2lba (tLARGE, tLBA *).`
   - `int g_idechs2chs (tIDECHS. tCHS *).`
   - `int g_idechs2lagre (tIDECHS. tLARGE *).`

- `int g_idechs2lba (tIDECHS. tLBA *).`
- `int a_lba2chs (tCHS geometry, tLBA, tCHS *).`
- `int a_lba2large (tLARGE geometry, tLBA, tLARGE *).`
- `int a_lba2idechs (tIDECHS geometry,  tLBA, tIDECHS *).`
- `int a_chs2lba (tCHS geometry, tCHS, tLBA *).`
- `int a_large2lba (tLARGE geometry, tLARGE, tLBA *).`
- `int a_idechs2lba (tIDECHS geometry, tIDECHS, tLBA *).`
- `int a_large2chs (tLARGE geometry1, tCHS geometry2, tLARGE, tCHS *).`
- `int a_large2idechs (tLARGE geometry1, tIDECHS geometry2, tLARGE, tIDECHS *).`
- `int a_chs2large (tCHS geometry1, tLARGE geometry2, tCHS, tLARGE *).`
- `int a_idechs2large (tIDECHS geometry1, tLARGE geometry2, tIDECHS, tLARGE *).`
- `int a_chs2idechs (tCHS geometry1, tIDECHS geometry2, tCHS, tIDECHS *).`
- `int a_idechs2chs (tIDECHS geometry1, tCHS geometry2, tIDECHS, tCHS *).`

Защита лабораторной работы

С использованием библиотеки функций необходимо разработать программу, выполняющую следующие действия:

- Предлагает пользователю ввести геометрию диска в формате IDECHS.
- Рассчитывает размер жесткого диска в ГБайтах и выводит его на экран.
- Предлагает пользователю ввести: размер требуемого раздела на диске, его тип и будет ли он активный (активным может быть только один раздел на диске !).
- На основании введённых данных рассчитывает строку в таблице разделов. Считается, что первый создаваемый пользователем раздел располагается, начиная с сектора 1 (LBA), второй – следом за ним, третий – следом за вторым и т.д.
- Формирование таблицы разделов прекращается, если пользователь ввёл 0 (ноль) как размер раздела или на диске больше не осталось свободного места.
- После ввода всей требуемой информации формируются таблицы разделов (основная и все расширенные) и выводятся на экран с указанием номера сектора, в котором будет записана каждая таблица.

Контрольные вопросы.

1. Основные этапы загрузки ПК на базе процессоров семейства Intel.
2. Зачем используется сигнал “RESET”?
3. Магнитные диски. Зачем используются. Устройство.
4. Магнитные головки чтения/записи. Типы. Зачем используются. Принцип ра-боты.
5. Привод магнитных головок. Типы приводов. Зачем используются.
6. Контроллер управления. Зачем используется.
7. Геометрия. Что это такое? Трансляция геометрии. Типы трансляции.
8. LBA адресация. Зачем используется. Перевод из LBA в CHSлог и наоборот.
9. Барьеры размеров дисков. Почему возникли? Какие присутствуют?
10. Этапы загрузки ПК.
11. Логическая организация винчестера. Разделы диска. Таблица разделов. Зачем используется. Структура.

КУРСОВАЯ РАБОТА

В рамках курсовой работы необходимо доработать модель Simple Computer так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице 1. Из пользовательских функций необходимо реализовать только одну согласно варианту задания (номеру вашей учетной записи). Для разработки программ требуется создать трансляторы с языков Simple Assembler и Simple Basic.

Обработка команд центральным процессором

Для выполнения программ моделью Simple Computer необходимо реализовать две функции:

- `int ALU (int command, int operand)` – реализует алгоритм работы арифметико-логического устройства. Если при выполнении функции возникла ошибка, которая не позволяет дальше выполнять программу, то функция возвращает -1, иначе 0;
- `int CU (void)` – обеспечивает работу устройства управления.

Обработку команд осуществляет устройство управления. Функция CU вызывается либо обработчиком сигнала от системного таймера, если не установлен флаг «игнорирование тактовых импульсов», либо при нажатии на клавишу “t”. Алгоритм работы функции следующий:

1. из оперативной памяти считывается ячейка, адрес которой храниться в регистре instructionCounter;
2. полученное значение декодируется как команда;
3. если декодирование невозможно, то устанавливаются флаги «указана неверная команда» и «игнорирование тактовых импульсов» (системный таймер можно отключить) и работа функции прекращается.
4. Если получена арифметическая или логическая операция, то вызывается функция ALU, иначе команда выполняется самим устройством управления.
5. Определяется, какая команда должна быть выполнена следующей и адрес её ячейки памяти заносится в регистр instructionCounter.
6. Работа функции завершается.

Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления. Пример программы на Simple Assembler:

```
00  READ    09      ; (Ввод A)
01  READ    10      ; (Ввод B)
02  LOAD    09      ; (Загрузка A в аккумулятор)
03  SUB     10      ; (Отнять B)
04  JNEG    07      ; (Переход на 07, если отрицательное)
05  WRITE   09      ; (Вывод A)
06  HALT    00      ; (Останов)
07  WRITE   10      ; (Вывод B)
08  HALT    00      ; (Останов)
09  =       +0000   ; (Переменная A)
10  =       +9999   ; (Переменная B)
```

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: sat файл.sa файл.o, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.o – результат трансляции.

Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Пример программы на Simple Basic:

```
10  REM Это комментарий
20  INPUT A
30  INPUT B
40  LET C = A – B
50  IF C < 0 GOTO 20
60  PRINT C
```

```
70 END
```

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, *, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

Оформление отчета по курсовой работе

Отчет о курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается диск с разработанным программным обеспечением. В пояснительную записку должны входить:

- титульный лист;
- полный текст задания к курсовой работе;
- реферат (объем ПЗ, количество таблиц, рисунков, схем, программ, приложений, краткая характеристика и результаты работы);
- содержание:
  - · постановка задачи исследования;
  - · блок-схемы используемых алгоритмов;
  - · программная реализация;
  - · результаты проведенного исследования;
  - · выводы;
- список использованной литературы;
- подпись, дата.

Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

Список литературы

1. Организация ЭВМ и систем. Практикум // С.Н. Мамойленко, Новосибирск: ГОУ ВПО «СибГУТИ», 2005 г., URL:
2. Архитектура компьютера. 4-е изд. // Э. Танненбаум. – СПб.: Питер, 2003.
3. Организация ЭВМ. 5-е изд. / К. Хамахер, З. Вранешич, С. Заки. – СПб.: Питер; Киев: Издательская группа BHV, 2003.
4. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: учебник для ВУЗов. – СПб.: Питер, 2004.