**Отчёт по самостоятельной работе 1**

**Выполнил Полыгалов Богдан ПИ-20-3**

# Задание 1

Принцип двоичного кодирования: все данные и команды кодируется с помощью битов (1 или 0). Двоичные последовательности разделяются на единицы, которые называют словами. Последовательность битов в слове, имеющая определённый смысл, называют полем. В числах выделяют поле знака и поле значащих разрядов. В командах выделяют поле кода операции и поле адресов. Преимущество двоичного кодирование заключается в том, что для реализации нужны элементы с двумя возможными состояниями, одно из которых будет представлять 1, а другое 0.

Принцип хранимой программы: хранение в оперативной памяти не только данных, но также программы, которой эти данные обрабатываются. Каждая команда программы занимает ячейку памяти и имеет свой адрес. Это позволяет производить с кодом программы такие же действия, как и с данными.

Принцип адресности памяти: оперативная память состоит структурно из ячеек, они же, в свою очередь, из битов (сегодня размер 1 ячейки – 8 бит). Каждая ячейка имеет свой адрес или по-другому номер. Нумерация начинается с 0. Процессору в любое время доступна любая ячейка, но за один раз можно прочитать или записать только целую ячейку.

Принцип иерархической организации памяти: память состоит из уровней: внутренний и внешний, получается своего рода иерархия. Также мы хотим, чтобы память была большой и быстрой, но такие требования противоречивы. Но иерархический способ организации позволяет иметь быстродействующую память небольшого объёма для данных и команд, а всё остальное хранится в более низком уровне – более объёмном, но менее быстродействующем.

# Задание 2

Основной алгоритм работы процессора:

1. Выбор команды
2. Запись адреса следующей команды в счётчик команд
3. Выполнение команды

И так происходит циклически, пока не будет команда остановки.

Счётчик адреса команд (регистр IP в Intel) указывает на ячейку памяти, в которой хранится следующая команда программы.

# Задание 3

Команда сравнения «CMP о1, о2» вычитает из о1 о2, но сами операнды при этом не изменяются, в этом заключается отличие от вычитания. Влияет на флаги процессора.

Флаги, а точнее регистр флагов – это специальный регистр, он состоит из битов и с их помощью отражает состояние процессора, а также может им управлять. Нужен для принятия решений. Главные флаги: SF, ZF, CF, OF, PF (из лекции). Это флаги состояния, которые отражают результат арифметических действий.

Флаги используются при условных переходах, поскольку команды условного перехода анализируют значение флагов. Например, команда условного перехода JZ выполнится, если флаг ZF будет равен 1, или команда JNZ выполнится, если флаг ZF будет равен 0.

# Задание 4

1. Непосредственная адресация. Используется непосредственно сама константа. Пример: 1. SUB AX, 4; 2. ADD BX, 5.
2. Регистровая адресация. Используются регистры. Пример: 1. SUB AX, BX; 2. ADD AX, BX.
3. Прямая адресация (абсолютная). Указывается адрес операнда, находящегося в памяти.

Пример:

1. a:= 100;

asm

MOV BX, [a]

1. ADD [a], AX
2. Неявная адресация. Операнд фиксирован и определяется командой. Пример: 1. MUL BX; 2. DIV BX.
3. Косвенная адресация. Адрес операнда находится в регистре. Пример: 1. ADD AX, [SI]; 2. SUB AX, [BX].
4. Косвенная со смещением. Пример: 1. ADD AX, [SI+2]; 2. SUB AX, [BX]+2.
5. Косвенная индексная адресация. Используются индексные регистры SI и DI. Пример: 1. MOV AX, [BX+DI]; 2. ADD AX, [BX+SI].
6. Стековая адресация. Адрес операнда находится в регистре указателя стэка SP.

Пример:

1. push 7

pop AX

1. jmp @1

@P1: mov BX,7

ret

@1: call @P1

# Задание 5

Выражение: 5\*7\*(3\*(4+5))-6:3

Выражение в польской системе: 57\*345+\*\*63:-

Вычисление с помощью стековой организации памяти, через | указывается содержимое стэка:

1. 5 ↑ | 5
2. 7 ↑ | 7 5
3. Из стэка достаётся 7 и 5, умножают, 35 ↑ | 35
4. 3 ↑ | 3 35
5. 4 ↑ | 4 3 35
6. 5 ↑ | 5 4 3 35
7. Из стэка достаётся 5 и 4, складывают, 9 ↑ | 9 3 35
8. Из стэка достаётся 9 и 3, умножают, 27 ↑ | 27 35
9. Из стэка достаётся 27 и 35, умножают, 945 ↑ | 945
10. 6 ↑ | 6 945
11. 3 ↑ | 3 6 945
12. Из стэка достаётся 3 и 6, делят, 2 ↑ | 2 945
13. Из стэка достаётся 2 и 945, вычитают, 943 ↑ | 943

# Задание 6

Используется стэк. Команда call сохраняет значение регистра EIP в стэк, это адрес на следующую команду, которая должна быть выполнена после call; а также передаёт управление подпрограмме (через изменение EIP). А затем команда ret извлекает этот адрес из стэка и возвращает управление на команду.

Каждая подпрограмма может вызывать другие подпрограммы или саму себя, поскольку при вызове используется адрес, по которому происходит переход. Адрес может быть и на другие подпрограммы, и на начало подпрограммы, из которой происходит вызов. Также из стэка можно достать адрес следующей команды после подпрограммы, так что произойдёт выход из подпрограммы.

Чтобы внутри подпрограммы получить адрес возврата из неё и занести его в регистр, не нарушая механизма возврата, нужно использовать команды pop и push:

jmp @1

@P1:

pop ebx

push ebx

ret

@1:

call @P1

Комментарий: pop ebx достаёт адрес из стэка и помещает в ebx, удаляя его в стэке. Push ebx добавляет его обратно в стэк, для завершения работы подпрограммы, сохраняя при этом его в ebx.

# Задание 7

AND ax,ax: сохраняет число в регистре AX, меняя флаги: SF 1 – если в AX отрицательное число, ZF 1 – если в AX 0, PF 1 если в двоичном представлении числа, хранящиеся в AX, чётное количество единиц.

XOR ax,ax: присваивает в AX 0, поскольку числа совпадают, флаги меняются: SF 0, потому что старший бит 0, ZF 1, поскольку все биты 0, PF 1 поскольку 0 единиц – чётное.

Команда XOR ax, 0FFFFh выполняет логическое отрицание числа в AX, равносильно команде NOT ax.

Таблицы истинности:

AND ax,ax:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

XOR ax,ax:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |
| 0 | 0 | 0 |

XOR ax, 0FFFFh

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

# Задание 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ax | флаг C | сдвиги | название операции | ax | флаг C |
| EAE0 | 1 | **SAR ax,1** | Арифметический сдвиг вправо | F570 | 0 |
| **SHR ax,1** | Логический сдвиг вправо | 7570 | 0 |
| **RCR ax,1** | Циклический сдвиг вправо через бит переноса | F570 | 0 |
| **ROR ax,1** | Циклический сдвиг вправо | 7570 | 0 |

Хотя разницу особо и не видно по таблице, но в работе команд она есть:

1. SAR: сдвигает все биты вправо, сохраняя старший бит, а младший сохраняется в флаг С;
2. SHR: сдвигает все биты вправо, в старший вписывается 0, а младший сохраняется в флаг С;
3. RCR: сдвигает все биты вправо, в старший вписывается значение флага С, а младший сохраняется в флаг С;
4. ROR: сдвигает все биты вправо, при этом младший бит идёт в старший и в флаг С.

# Задание 9

{$asmmode intel}

{$output format asw}

var c: integer;

begin

asm

MOV ax, -100

AND ax, 0FFFFh

JGE @1

NOT ax

INC ax

@1:

MOV c,ax

end;

Writeln(c);

Readln();

end.

Комментарий:

Сначала с помощью команды AND определяем знак числа: если число отрицательное, то после команды регистр SF сменится на 1, если положительное останется 0. Также было выбрано число FFFF, чтобы сохранить число, которое находится в регистре AX.

Затем идёт условный переход, который работает, если SF = OF, OF у нас сохраняет 0, так как команда AND на него не влияет. Когда число положительное, то 0 = 0 и срабатывает переход на метку, после которой программа заканчивает работу. Если число отрицательное, то переход не срабатывает и выполняются команды NOT и INC, c помощью которых меняется знак у числа.

# Задание 10

Программа сравнивает числа в регистрах BX и CX, затем в AX присваивается меньшее число, или если числа равны, то само число.

Явно неправильный переход находится в метке @1: JMP @1. Так программа зацикливается. Надо исправить на JMP @3.

Чтобы уменьшить количество переходов можно заменить JL и JE на JLE.

CMP bx,cx

JLE @1

MOV ax,cx

JMP @2

@1:

MOV ax,bx

@2: