Симулятор микрокомпьютера с архитектурой фон Неймана

Симулятор находится на web-странице по адресу http://prog.tversu.ru/assembler-simulator/index.html.

Возможности симулируемого микрокомпьютера:

- микрокомпьютер является 8-битным;
- имеются 4 однобайтных регистра общего назначения: A,B,C,D;
- имеются 2 специальных регистра: IP – счётчик команд и SP – указатель стека (адреса, на которые указывают эти регистры, подсвечиваются соответствующим цветом);
- имеются 3 флага: Z – ноль (англ. zero), C – перенос (англ. carry), F – процессор остановлен из-за ошибки (англ. fault) ;
- компьютер имеет память размером 256 байт;
- последние 24 байта памяти (начиная с адреса 232) выводятся в виде символов в окне Output, организованном как текстовый экран размером 8x3 символа;
- компьютер программируется на языке ассемблера, близкого к реальным ассемблерам;
- значения в памяти и регистрах могут отображаться в десятичном и шестнадцатеричном виде, переключение осуществляется ссылкой View: Hex / Decimal под окном памяти.

Работа с симулятором

При открытии web-страницы с симулятором, в окне исходного кода содержится пример программы. Текст этого примера можно удалить и заменить своей программой.

После ввода текста программы нужно нажать кнопку Assemble. При этом код программы будет преобразован в машинный код и размещен в памяти. В случае наличия синтаксических ошибок, строка с ошибкой будет выделена и вверху появится соответствующее сообщение.

Управление работой симулятора осуществляется кнопками, находящимися в левой части заголовка окна.

Для пошагового выполнения программы используйте кнопку ▶▶Step. Будет выполнена одна инструкция.

Последовательное выполнение всех инструкций программы запускается кнопкой ▶Run.

Кнопка Reset сбрасывает симулятор в начальное состояние (не удаляя текущую программу).

Если в процессе работы произойдёт ошибка (например, процессор прочитает некорректный машинный код), то будет выведено соответствующее сообщение, флаг F примет значение Истина (англ. True) и работа симулятора остановится.

Язык ассемблера для симулятора

Каждая инструкция программы должна занимать отдельную строку, и включает компоненты, показанные на рисунке 1.

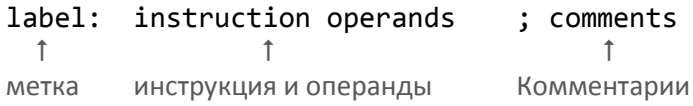

Рис. 1. Синтаксис команды ассемблера

Метка является не обязательной, и используется для команд или байтов данных, на которые потребуется ссылаться из других частей программы. Метка должна начинаться с буквы или точки и заканчиваться двоеточием.

Комментарий является не обязательным и используется для пояснения сути программы. Он начинается с двоеточия и продолжается до конца строки. Ассемблер игнорирует содержание комментария. В программе допускаются строки, содержащие только комментарий.

Записанные на языке ассемблера инструкции транслируются в машинные коды и размещаются в памяти одна за другой, в том порядке, в котором они указаны в программе. Размещение программы начинается с адреса 0. Все инструкции, кроме DB, требуют по одному байту для кода операции и каждого аргумента. Симулятор имеет режим подсветки границ инструкций, который переключается ссылкой Instructions: Hide / Show под окном памяти.

Операнды

В качестве операндов в инструкциях может выступать регистр общего назначения, указатель стека, константа или адрес. Указатель стека может быть использован только с инструкциями MOV, ADD, SUB, INC, DEC и CMP.

Имя регистра общего назначения указывается буквой: A, B, C или D.

Указатель стека указывается сочетанием букв SP.

Запись констант

Цифровые константы от 0 до 255 могут быть записаны следующими способами:

- в десятичной системе: `200` или `200d`;
- в шестнадцатеричной системе: `0xA4`;
- в восьмеричной системе: `0o48`;
- в двоичной системе: `101b`.

Используя инструкцию DB можно задать в памяти символьные константы:

- отдельный символ, используя одинарные кавычки: `'A'`;
- последовательность символов, используя двойные кавычки: `"Hello!"`.

Запись адресов

Адрес в оперативной памяти может быть указан следующими способами:

- как константа – число от 0 до 255 в квадратных скобках: `[100]`;
- с помощью метки – ассемблер заменит имя метки соответствующим ей адресом [label];
- косвенная адресация по значению в регистре – имя регистра, указанное в квадратных скобках, позволяет сослаться на ячейку памяти, адрес которой записан в этом регистре: `[A]`;
- косвенная адресация со смещением от значения в регистре: в этом случае процессор берёт значение из регистра, увеличивает или уменьшает его на значение смещения и использует полученное значение в качестве адреса: [B-5].

Косвенная адресация может быть использована с регистрами общего назначения или указателем стека (SP). Смещение может принимать значения от -16 до +15.

Набор инструкций

DB

Инструкция DB служит для определения переменных в памяти. В результате трансляции этой команды в память записывается значение её аргумента. Чтобы на это значение можно было сослаться используйте метки.

Примеры:

```
DB 80      ; следующий байт получит значение 80
DB 'A'     ; следующий байт получит значение кода буквы A – 0x41.
DB "text"  ; в следующие 4 байта будут записаны коды букв t,e,x,t.
```

HLT

Инструкция HLT останавливает работу процессора. Перед повторным запуском необходимо нажать кнопку Reset, чтобы сбросить регистр IP в начальное значение.

MOV

Инструкция MOV перемещает значения между регистрами или между регистрами и памятью. Это единственная инструкция, позволяющая непосредственно изменить значение в памяти.

Первый аргумент указывает, куда должно быть помещено значение. Второй аргумент – откуда его взять.

Допустимые комбинации:

```
MOV регистр, регистр
MOV регистр, адрес
MOV регистр, константа
MOV адрес, регистр
MOV адрес, константа
```

Примеры:

```
MOV A,0x10 ; поместить в регистр A значение 0x10
MOV [B],A  ; поместить значение из регистра A в ячейку памяти,
           ; номер которой записан в регистре B
```

Арифметические операции

Увеличение и уменьшение

Для увеличения и уменьшения значения регистра на 1 используются инструкции INC и DEC соответственно.

Эти инструкции имеют один операнд – регистр, значение которого будет изменено.

Во время выполнения изменяются флаги Z и C.

Пример:

```
INC C      ; увеличить значение в регистре C на 1.
```

Сложение и вычитание

Для сложения используется инструкция ADD, для вычитания – SUB.

Первым операндом обязательно должен быть регистр, вторым может быть регистр, адрес в памяти или константа. После выполнения операции результат помещается в регистр, указанный первым оператором.

Эти инструкции модифицируют флаги Z и C.

Примеры:

```
ADD A,10   ; A=A+10
SUB B,C    ; B=B-C
```

Умножение и деление

Инструкция умножения MUL умножает, а инструкция деления DIV делит значение в регистре A на значение аргумента. В качестве аргумента могут выступать регистр, адрес или константа.

Эти инструкции модифицируют флаги Z и C.

Примеры:

```
MUL B      ; A=A*B
MUL 3      ; A=A*3
```

Логические операции

Поддерживаются следующие логические операции:

- AND – и;
- OR – или;
- XOR – исключающее или;
- NOT – отрицание (инвертирование битов).

В качестве первого операнда всегда выступает регистр. В качестве второго операнда у инструкций AND, OR и XOR может выступать регистр, адрес или константа. Результат операции помещается в регистр, который был указан в качестве первого аргумента.

Инструкции модифицируют флаги Z и C.

Примеры:

```
AND B,C     ; B=B (побитовое и) C
NOT D       ; инвертировать биты регистра D
```

Сдвиги

Реализованы инструкции сдвига влево SHL и вправо SHR. В качестве первого аргумента выступает регистр, значение в котором сдвигается. Размер сдвига задаётся вторым аргументом и может быть регистром, адресом или константой.

Инструкции модифицируют флаги Z и C.

Пример:

```
SHL A,2     ; сдвинуть биты регистра A на 2 бита влево.
```

Сравнение

Инструкция CMP вычитает свои аргументы и оставляет их без модификации. Единственным следствием выполнения этой инструкции является модификация флагов Z и C. Обычно эта инструкция используется для сравнения чисел перед операцией условного перехода.

Пример:

```
CMP B, 10  ; проверить, равно ли значение в регистре B десяти
JNZ .loop  ; переход на метку .loop если B не равно 10.
```

Переходы

Безусловный переход

Инструкция JMP заставляет перейти к выполнению инструкции по заданному адресу.

Пример:

```
JMP done    ; перейти к выполнению инструкции с меткой done
```

Условные переходы

Инструкции условного перехода осуществляют переход по заданному адресу, только если выполнено соответствующее условие.

В качестве условий выступают определённые комбинации значений флагов Z (ноль) и C (перенос). Значение этих флагов зависит от последней выполненной арифметической операции. Используя перед условным переходом операции вычитания SUB или сравнения CMP (если не желательно повреждать аргументы) можно реализовать различные операции сравнения.

Если условие не выполнено, управление будет передано к следующей по порядку инструкции.

Поддерживаемые инструкции безусловного перехода представлены в таблице 1. Обратите внимание, что для всех машинных инструкций (вариантов условий) имеется несколько эквивалентных имён. Конкретный синоним может использоваться, для того, чтобы указать, какой именно смысл имела операция сравнения.

Таблица 1. Инструкции условного перехода

Инструкция	Описание	Условие	Синонимы
JC	Jump if carry	C=TRUE	JB, JNAE

	Переход, если перенос		
JNC	Jump if no carry Переход, если нет переноса	C=FALSE	JNB, JAE
JZ	Jump if zero Переход, если ноль	Z=TRUE	JE
JNZ	Jump if no zero Переход, если не ноль	Z=FALE	JNE
JA	Переход, если >	C=FALSE и Z=FALSE	JNBE
JNBE	Переход, если не верно <=	C=FALSE и Z=FALSE	JA
JAE	Переход, если >=	C=FALSE	JNC, JNB
JNB	Переход, если не верно <	C=FALSE	JNC, JAE
JB	Переход, если <	C=TRUE	JC, JNAE
JNAE	Переход, если не верно >=	C=TRUE	JC, JB
JBE	Переход, если <=	C=TRUE или Z=TRUE	JNA
JNA	Переход, если не верно >	C=TRUE или Z=TRUE	JBE
JE	Переход, если =	Z=TRUE	JZ
JNE	Переход, если не верно =	Z=FALSE	JNZ

Пример:

```
CMP A,B
JA greater ; переход произойдет, если A>B
```

Работа со стеком

Специальный регистр SP указывает на первый свободный элемент в стеке. При добавлении элементов в стек значение уменьшается. В начале работы SP указывает на последний байт перед областью вывода – 231.

Таким образом, стек начинается с доступного конца памяти и увеличивается к её началу, тогда как программа начинается с начала памяти и растёт в направлении её конца (рисунок 2). Для корректной работы стек не должен дойти до области памяти, использованной под программу.

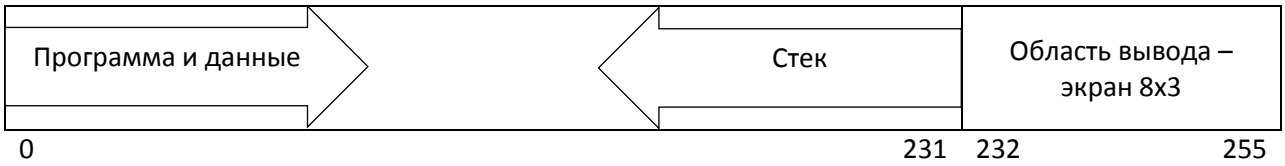

Рис. 2. Распределение оперативной памяти

Для помещения значения в стек используется инструкция PUSH. В качестве аргумента можно использовать регистр, адрес или константу. После выполнения инструкции значение регистра SP уменьшается.

Чтобы извлечь значение из стека используется инструкция POP. Аргументом инструкции POP может являться только регистр. Инструкция увеличивает значение регистра SP.

Необходимо следить, чтобы при работе программы число вызовов инструкции POP не превышало числа вызова инструкции PUSH – не следует брать из стека больше, чем туда положили. Иначе значения будут «вываливаться» из области памяти экрана.

Примеры:

```
PUSH 42    ; поместить в стек константу 42
; … программа продолжается …
POP B      ; извлечь из стека значение и поместить в регистр B
```

Работа с подпрограммами

Инструкция CALL используется для вызова подпрограммы. При её выполнении:

- адрес следующей за CALL инструкцией помещается в стек;
- управление передаётся инструкции по заданному адресу.

Для возврата из подпрограммы следует использовать инструкцию RET. Она извлекает из стека адрес для возврата, ранее сохранённый инструкцией CALL и выполняет переход по этому адресу.

Инструкция RET будет корректно работать только в том случае, если работа со стеком внутри неё была сбалансирована – число помещённых туда элементов (с помощью PUSH или CALL) равно числу извлечённых элементов (с помощью POP или RET). Иначе из стека будет извлечено неправильное значение, и управление будет передано по неправильному адресу.

Пример:

```
CALL my_function            ; вызов функции my_function
; … программа продолжается …

my_function:
                ;… код функции …
                RET         ; возврат из функции
```