## Пример 1. Логические операции

С точки зрения программиста логические операции практически совпадают с арифметическими операциями. Принципиальное отличие состоят в том, что логические операции не создают выполнение (кроме операций сдвига), и вам не нужно беспокоиться об отрицательных числах. Логические операции называются побитовыми, поскольку они действуют на отдельные биты. Основные или фундаментальные логические операции:

| | | |
|---|---|---|
| **NOT** | `Invert bits` | $c_i = \overline{a}_i$ |
| **AND** | `Logical and` | $c_i = a_i \cdot b_i$ |
| **OR** | `Logical OR` | $c_i = a_i + b_i$ |

Производными логическими операциями, которые могут быть выражены в терминах фундаментальных операций, являются (это не исчерпывающий список):

| | | |
|---|---|---|
| **XOR** | `Exclusive OR` | $c_i = \overline{a}_i \cdot b_i + a_i \cdot \overline{b}_i$ |
| **NAND** | `NOT AND` | $c_i = \overline{a_i \cdot b_i}$ |
| **NOR** | `NOT OR` | $c_i = \overline{a_i + b_i}$ |

Операциями сдвига иногда могут быть группы с логическими операциями, а иногда - нет. Это связано с тем, что эти операции не являются фундаментальными Булевыми операциями, а являются операциями над битами. Операция сдвига перемещает все биты слова на одно или несколько позиций влево или вправо. Типичные операции сдвига:

| | | |
|---|---|---|
| **LSL** | `Logical shift left` | Shift the bits left. A 0 enters at the right hand position and the bit in the left hand position is copied into the carry bit. |
| **LSR** | `Logical shift right` | Shift the bits right. A 0 enters at the left hand position and the bit in the right hand position is copied into the carry bit. |
| **ROL** | `Rotate left` | Shift the bits left. The bit shifted out of the left hand position is copied into the right hand position. No bit is lost. |
| **ROR** | `Rotate right` | Shift the bits right. The bit shifted out of the right hand position is copied into the left hand position. No bit is lost. |

Система команд некоторых микропроцессоров включает и другие логические операции, которые не являются обязательными и могут быть синтезированы с использованием других операций.

| | |
|---|---|
| `Bit Set` | The bit set operation allows you to set bit `i` of a word to 1. |
| `Bit Clear` | The bit clear operation allows you to clear bit `i` of a word to 0. |
| `Bit Toggle` | The bit toggle operation allows you to complement bit *`i`* of a word to its complement. |

### Логические операции ARM

Немногие микропроцессоры реализуют все вышеперечисленные логические операции. Некоторые микропроцессоры реализуют специальные логические операции. Логическими операциями ARM являются:

| | | |
|---|---|---|
| ***MVN*** | **`MVN r0,r1`** | $r0 = \overline{r1}$ |
| ***AND*** | `AND` **`r0,`**`r1,r2` | $r0 = r1 \cdot r2$ |
| ***ORR*** | `OR` **`r0,`**`r1,r2` | $r0 = r1 + r2$ |
| ***EOR*** | `XOR` **`r0,`**`r1,r2` | $r0 = r1 \oplus r2$ |
| ***BIC*** | `BIC` **`r0,`**`r1,r2` | $r0 = r1 \cdot \overline{r2}$ |
| ***LSL*** | `MOV` **`r0,`**`r1,LSL r2` | `r1` is shifted left by the number of places in `r2` |
| ***LSR*** | `MOV` **`r0,`**`r1,LSR r2` | `r1` is shifted right by the number of places in `r2` |

Двумя необычными командами являются MVN (перемещение с отрицанием) и BIC (сброс бита). Инструкция с отрицанием перемещения действует скорее как инструкция перемещения (MOV), за исключением того, что биты, в этом случае, инвертируются. Обратите внимание, что биты в исходном регистре остаются без изменения. Команда BIC очищает биты первых операндов, когда установлены биты операнда назначения. Эта операция эквивалентна И между первым и отрицательным вторым операндом. Например, 8-битная операция `BIC r0,r1,r2` (с `r1=00001111` и `r2=11001010`) приведет к результату `r0=11000000`.

**Задача:**
Выполним простую Булеву операцию для вычисления побитовой операции $F = A \cdot B + \overline{C \cdot D}$.
Предположим, что A, B, C, D находятся в `r1, r2, r3, r4`, соответственно.

Фрагмент кода:

```
AND r0,r1,r2        ; r0 = A·B
AND r3,r3,r4        ; r3 = C·D
MVN r3,r3           ; r3 = NOT(C·D)
ORR r0,r0,r3        ; r0 = A·B + NOT(C·D)
```

На рис. Example 1 показано состояние системы после выполнения программы.
Следует отметить некоторые нюансы. Во-первых, мы использовали псевдо команду `LDR r1,=2_00000000111111101010101011110000`, чтобы загрузить двоичный литерал в регистр `r1`.
ARM не может загрузить 32-битную константу в одной команде (так как сама инструкция имеет ширину в 32 бита).
Формат `LDR r1,=` генерирует относительную загрузку счетчика команд и ассемблер автоматически помещает константу в соответствующее место в памяти.
Обратите внимание на формат числа по основанию 2. Это _2xxxx..x, где _2 указывает на двоичное представление, а x…x являются битами двоичного числа.

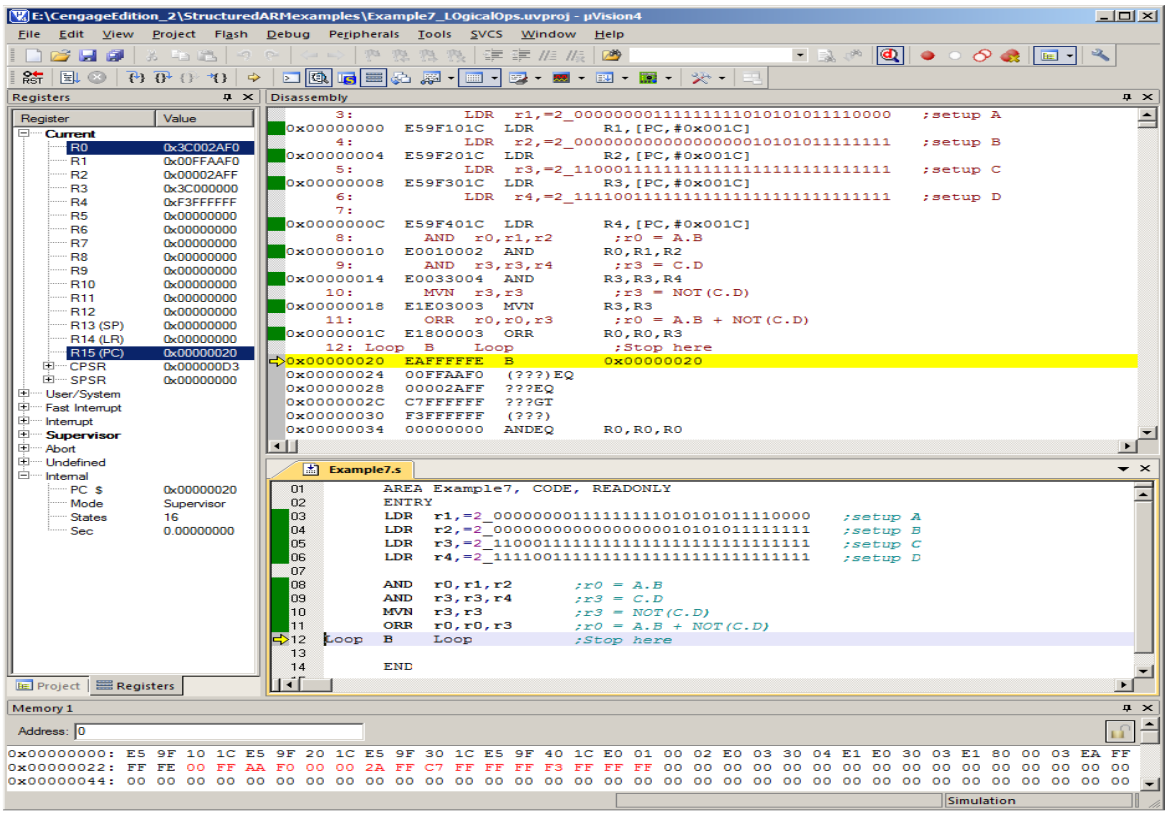

Рис. Example 1. Состояние системы после выполнения программы.