# 基礎コンピュータ工学
# 第５章 機械語プログラミング
# （パート１０）

# 論理演算

「*2.8* コンピュータの基本回路」で学んだ論理演算を再確認.

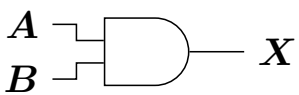
## 論理積（*AND*）

| 入力 | | 出力 |
|---|---|---|
| $A$ | $B$ | $X$ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND の真理値表

$$X = A \cdot B$$

AND の論理式




AND の回路記号

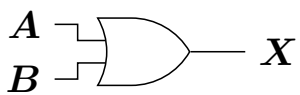
## 論理和（*OR*）

| 入力 | | 出力 |
|---|---|---|
| $A$ | $B$ | $X$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR の真理値表

$$X = A + B$$

OR の論理式




OR の回路記号

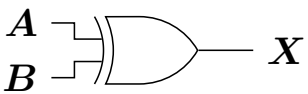
## 排他的論理和（*XOR*）

| 入力 | | 出力 |
|---|---|---|
| $A$ | $B$ | $X$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XOR の真理値表

$$X = A \oplus B$$

XOR の論理式




XOR の回路記号

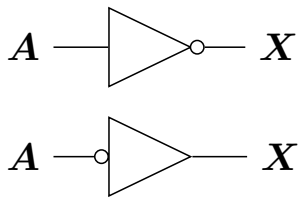
## 否定（*NOT*）

| 入力 | 出力 |
|---|---|
| $A$ | $X$ |
| 0 | 1 |
| 1 | 0 |

NOT の真理値表

$$X = \overline{A}$$

NOT の論理式

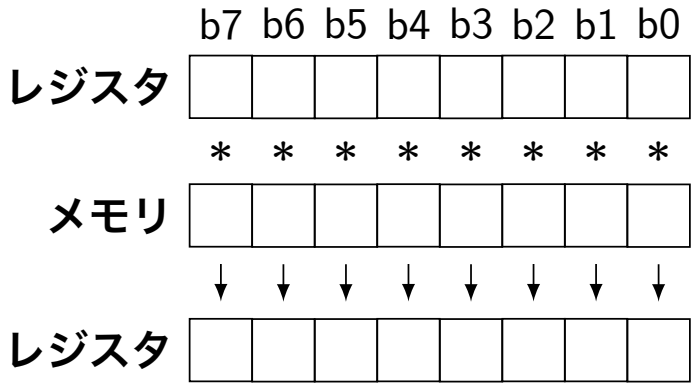



NOT の回路記号

# 論理演算命令

**論理演算を行う *TeC* の命令**

8 ビットデータを単位に，ビット毎の論理演算を行う．

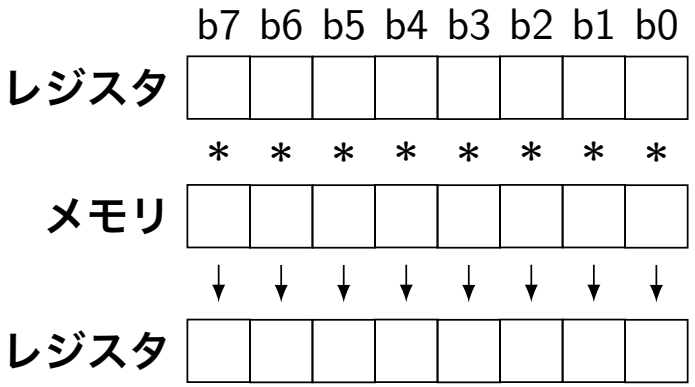



次の 3 種類がある（NOT はない）．

- **論理積（*AND*）**
- **論理和（*OR*）**
- **排他的論理和（*XOR*）**

# AND（Logical AND）命令（論理積）

レジスタ値とメモリ値のビット毎の**論理積**を計算し，結果をレジスタに格納する．




**C フラグ** 常に 0 になる．

**S フラグ** 結果が負（MSB が 1）なら 1，それ以外は 0 になる．

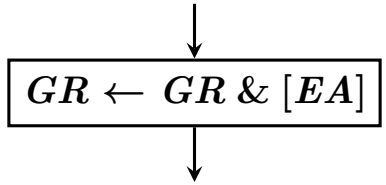
**Z フラグ** 結果がゼロなら 1，それ以外は 0 になる．

**ニーモニック：** `AND GR,EA`　　（GR ← GR & [EA]）

**命令フォーマット：** 2 バイトの長さを持つ．

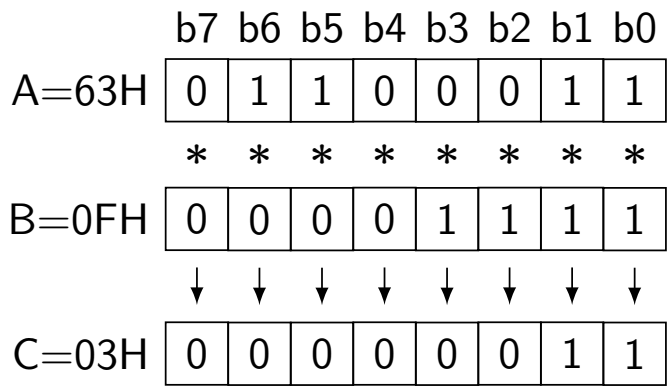
| 第 1 バイト | | 第 2 バイト |
|---|---|---|
| OP | GR XR | |
| $0110_2$ | *GR XR* | *aaaa aaaa* |

**フローチャート：** Java の演算子を流用する.




**使用例：** A 番地のデータと B 番地のデータのビット毎の論理積を計算し，C 番地に格納するプログラムの例を示す.

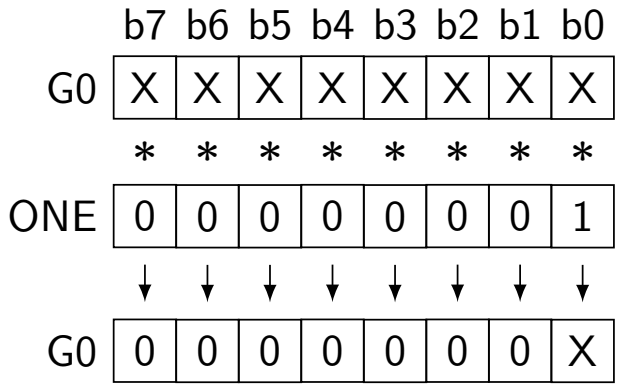
| 番地 | 機械語 | ラベル | ニーモニック | |
|---|---|---|---|---|
| 00 | 10 07 | | LD | G0,A |
| 02 | 60 08 | | AND | G0,B |
| 04 | 20 09 | | ST | G0,C |
| 06 | FF | | HALT | |
| 07 | 63 | A | DC | 63H |
| 08 | 0F | B | DC | 0FH |
| 09 | 00 | C | DS | 1 |



# AND 命令の応用（1）

**特定のビットがゼロか判定する.**
AND の結果が $00_{16}$ かどうかで判断できる.
次は最下位ビット（LSB）を調べる例.

| ラベル | ニーモニック |
|---|---|
| | ... |
| | AND G0,ONE |
| | JZ L1 |
| | ... |
| L1 | ... |
| | ... |
| ONE | DC 01H |

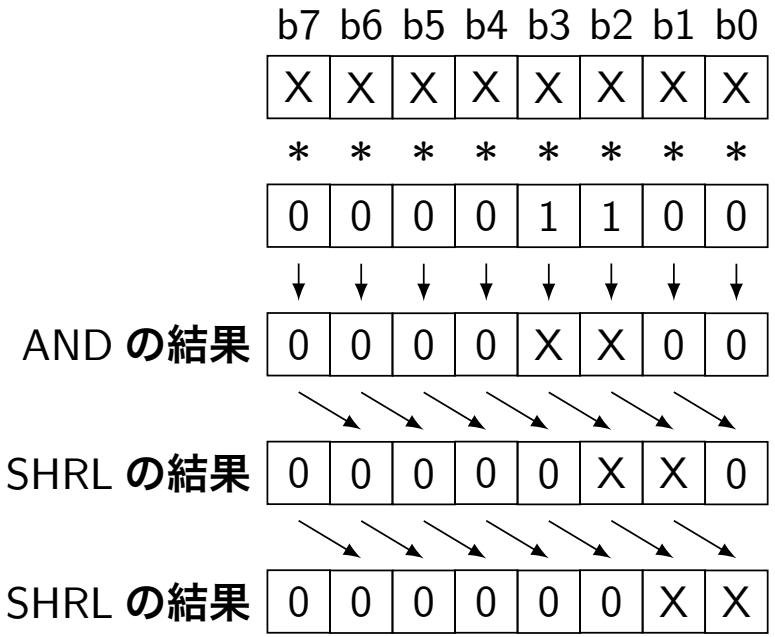



LSB : Least Significant Bit（最下位ビットのこと, P.10 参照）

# AND 命令の応用（2）

**特定のビットを右詰めで取り出す.**
（$b_3$, $b_2$ を右詰めで取り出す.）

| ラベル | ニーモニック | |
|---|---|---|
| | ... | |
| | AND | GO,MSK |
| | SHRL | GO |
| | SHRL | GO |
| | ... | |
| MSK | DC | 0CH |



# OR（Logical OR）命令（論理和）

レジスタ値とメモリ値のビット毎の**論理和**をレジスタに格納する.

**C フラグ** 常に 0 になる.
**S フラグ** 結果が負（MSB が 1）なら 1，それ以外は 0 になる.
**Z フラグ** 結果がゼロなら 1，それ以外は 0 になる.

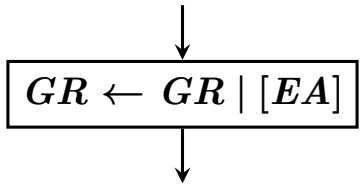
**ニーモニック：** `OR GR,EA`　　（GR ← GR | [EA]）

**命令フォーマット：** 2 バイトの長さを持つ.

| 第 1 バイト | | 第 2 バイト |
|---|---|---|
| OP | GR XR | |
| $0111_2$ | *GR XR* | *aaaa aaaa* |

MSB : Most Significant Bit（最上位ビットのこと, P.10 参照）

**フローチャート：** Java の演算子を流用する．




**応用：** G0 の上位 4 ビットを全部 1 にする．

| ラベル | ニーモニック | |
|---|---|---|
| | ... | |
| | LD | G0,DATA |
| | OR | G0,MSK |
| | ... | |
| DATA | DC | 0AAH |
| MSK | DC | 0F0H |

| | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | + | + | + | + | + | + | + | + |
| | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| OR **の結果** | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

*16* **進数の表記：**ラベルと区別が付くように注意！ (P.44 参照)

# XOR（Logical XOR）命令（排他的論理和）

レジスタ値とメモリ値のビット毎の**排他的論理和**をレジスタに格納する．

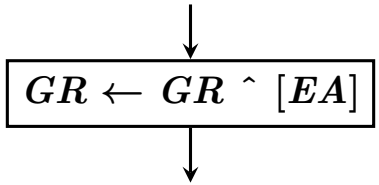
**C フラグ** 常に 0 になる．
**S フラグ** 結果が負（MSB が 1）なら 1，それ以外は 0 になる．
**Z フラグ** 結果がゼロなら 1，それ以外は 0 になる．

**ニーモニック：** `XOR GR,EA`　　（GR ← GR ^ [EA]）

**命令フォーマット：** 2 バイトの長さを持つ．

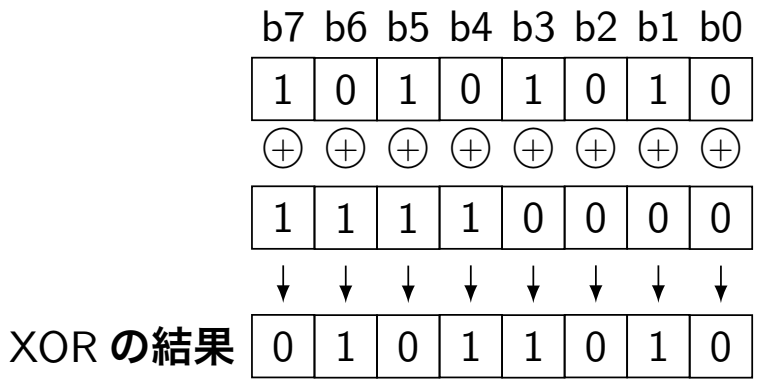
| 第 1 バイト | | 第 2 バイト |
|---|---|---|
| OP | GR XR | |
| $1000_2$ | *GR XR* | *aaaa aaaa* |

**フローチャート：** Java の演算子を流用する.




**応用：** G0 上位 4 ビットの 1/0 を入れ替える（ビット反転する）.

| ラベル | ニーモニック | |
|---|---|---|
| | ... | |
| | LD | G0,DATA |
| | XOR | G0,MSK |
| | ... | |
| DATA | DC | 0AAH |
| MSK | DC | 0F0H |



# まとめ

## 学んだこと

- ビット毎の論理演算命令
- TeC は次の演算命令を持っている.
  1. **論理積（*AND*）命令**
  2. **論理和（*OR*）命令**
  3. **排他的論理和（*XOR*）命令**

## 演習

- TeC には NOT 命令が無い.
  NOT 命令があったとすると，どんな計算をする命令になるか？
- NOT 命令の代用となる命令を考えなさい.
- 値が奇数か偶数か判定する方法を考えなさい.
- 8 で割った余りを計算する方法を考えなさい.