計算機論理設計

1. 計算機構成概要

一色 剛

工学院情報通信系

isshiki@ict.e.titech.ac.jp

講義の目的

■ 計算機構成 : ソフトウェア(SW)とハードウェア(HW)

- SW : 命令セット、機械語、アセンブリ言語
- HW : メモリ、レジスタ、演算器、バス

→ 計算機を実現するためのSW/HW構成要素の理解

■ 計算機論理設計手法

- 「論理回路理論」: 論理代数、論理最適化、状態遷移表からの論理式導出、状態数最小化 → 与えられた論理関数や状態遷移関数から論理回路を設計する方法を習得
- 本講義 : SW/HW構成・仕様から、必要な論理関数や状態遷移関数を導出する方法を習得

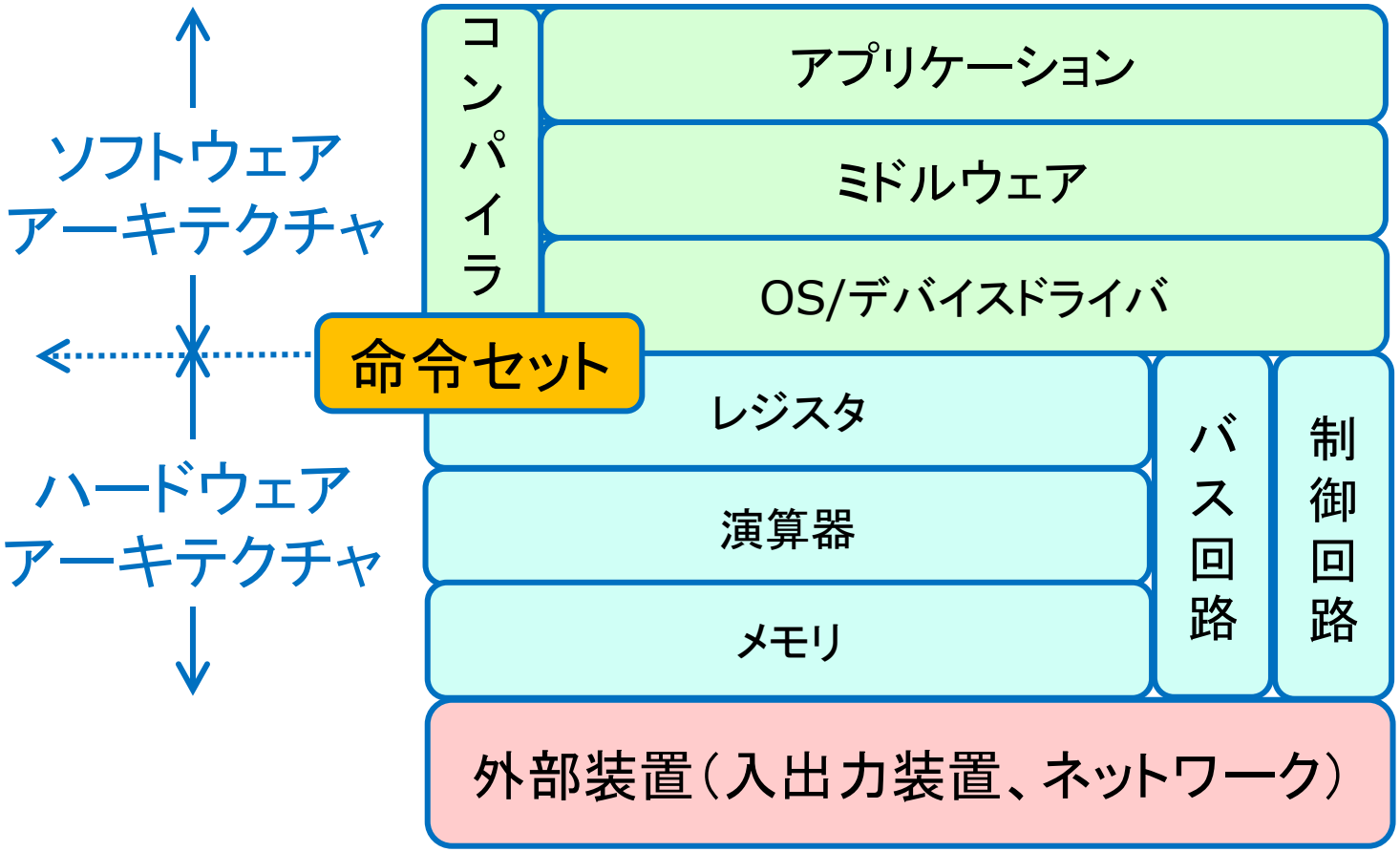
→ レジスタ転送記述に基づいた論理回路設計手法の理解

→ ハードウェア記述言語の習得 : レジスタ転送を表現するプログラミング言語

講義の流れ

- **計算機概論**
  - 計算機SW/HW構成、データ表現と演算回路
- **SW構成**
  - 命令セット、機械語、アセンブリプログラミング
- **HW構成**
  - 命令実行制御、レジスタ転送記述、入出力、割込機構
  - 回路設計 : レジスタ、バス、算術論理演算器
- **ハードウェア記述言語**
  - 論理回路記述、論理シミュレーション
  - モジュール、階層化、信号、論理演算式
  - 組合せ回路・順序回路、演算回路記述、同期式回路設計

計算機構成


ソフトウェア
アーキテクチャ
ハードウェア
アーキテクチャ
コンパイラ
アプリケーション
ミドルウェア
OS/デバイスドライバ
命令セット
レジスタ
演算器
メモリ
バス回路
制御回路
外部装置（入出力装置、ネットワーク）

ソフトウェアアーキテクチャ

- 命令セット：ソフトウェアとハードウェアの接点
  - 計算機の基本動作を定義した手続き（命令）の集合
  - SW：命令セットは実行可能形式（実行バイナリ）を定義
  - HW：命令セットは論理回路設計の「仕様」
- システムソフトウェア：
  - 計算機実行を補助：オペレーティングシステム、コンパイラ、ミドルウェア、デバイスドライバ
- アプリケーションソフトウェア：
  - 用途別に準備した情報処理プログラム

→限られた命令セットを組合せることで複雑な機能を実現するソフトウェア構成法を本講義で考える

ハードウェアアーキテクチャ

- **メモリ :主記憶装置**
  - アドレス（M bits）、データ（N bits） → (2^M x N) bits容量
  - 任意のメモリ番地にアクセス → Random Access Memory
- **レジスタ : 一時記憶装置**
  - 演算結果保存、プログラムカウンタ : フリップフロップで実現
- **演算回路: 情報処理を実行する論理回路**
  - 算術演算（四則演算）、論理演算、シフト演算
- **制御回路: メモリ・レジスタ・演算回路の制御**
  - プログラム命令列に従って計算機全体を駆動制御する

2進データ表現

- 符号なし整数 (unsigned)

$$value = \sum_{i=0}^{N-1} b_i \cdot 2^i \quad (b_i : i^{\text{th}} \text{ bit})$$

- 符号付き整数 (signed : 2の補数)

$$value = -b_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i \cdot 2^i$$

  – 最上位ビット (most significant bit: MSB) : “sign” bit

sign = 1 : 負整数

sign = 0 : 非負整数

2進	unsigned	signed
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

2進数の計算法
(Computer Arithmetic)

- 2進数の計算は10進数の「筆算」と同じ原理
- 2進数の各桁は0か1なので、計算機に好都合

→ 2進数計算は論理回路の主要な機能
(computer arithmetic)

```
  7   00111
+ 5   00101
 12   01100
```

```
  7   00111
- 5   00101
  2   00010
```

```
   12        01100
x  13      x 01101
   36        01100
  12        00000
  156      01100
          01100
         00000
         010011100
```

```
      22          00010110
5 ) 114     101 ) 01110010
    10            101
     14           01000
     10             101
      4             0111
                     101
                      100
```

2進数加算

- 各ビットで (最下位ビット（左）から最上位ビット（右）へ):
  - 1の個数(“one-count”)を2進数で表現
  - “out-count”の上位ビット : 次のビットのキャリー（桁上げ）

carry-bit

```
      0        1        1        1        0
  00111    00111    00111    00111    00111
+ 00101  + 00101  + 00101  + 00101  + 00101
-------  -------  -------  -------  -------
     10       10       11       01       00
-------  -------  -------  -------  -------
      0       00      100     1100    01100
```

sum-bit

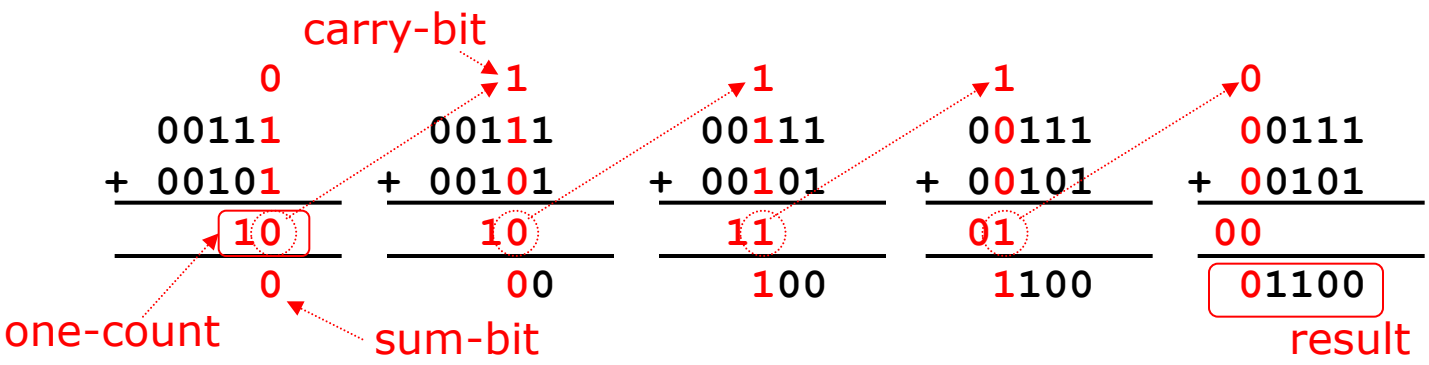
result

one-count

```
00111 + 00101 = 01100
  7   +   5   =  12
```

1ビット加算器 (Full Adder)

- 3つのビット入力の1の個数を計算する論理回路



3-bit input

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

one-count (2-bit output)

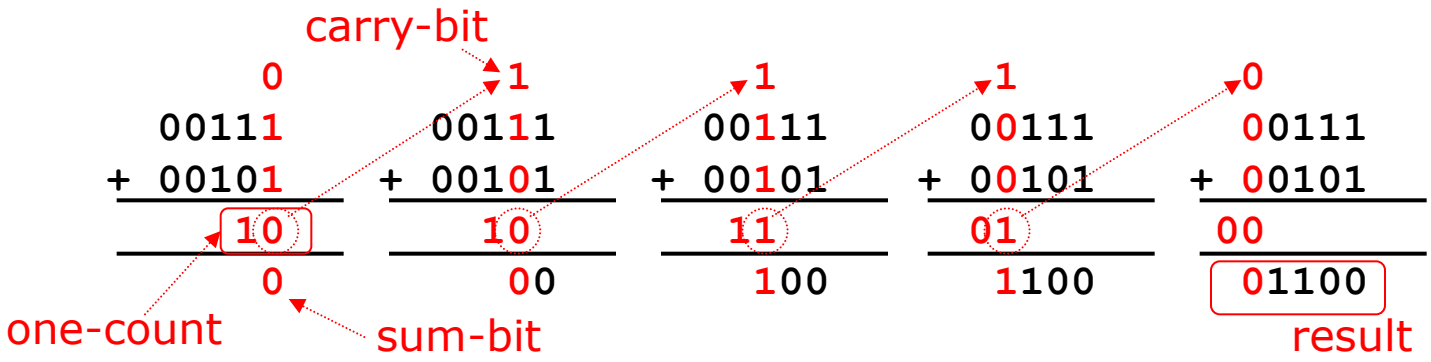
$$c = x \cdot y + y \cdot z + z \cdot x$$

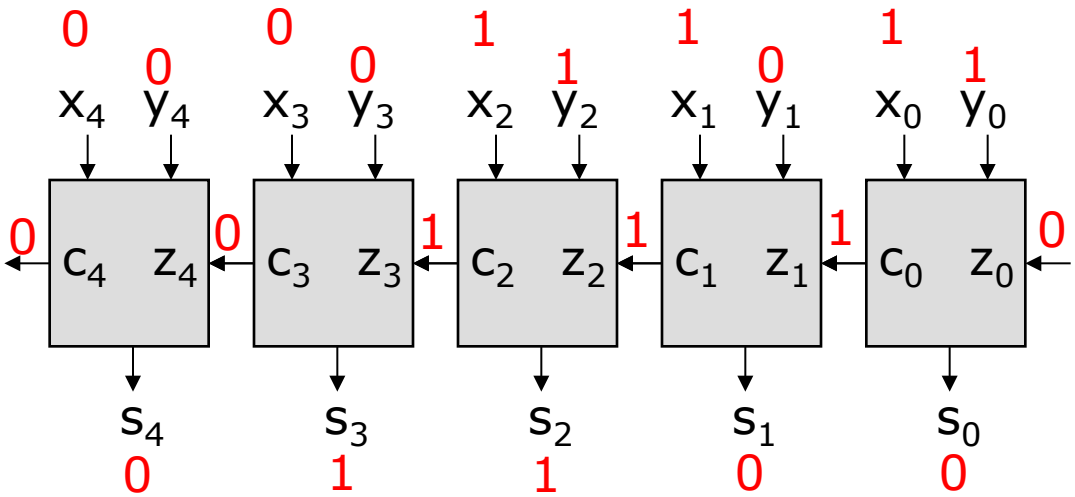
$$s = x \oplus y \oplus z$$

$$= (\sim x \ \sim y \ z) + (\sim x \ y \ \sim z) + (x \ \sim y \ \sim z) + (x \ y \ z)$$

Nビット加算器

- Nビット加算器：N個の1ビット加算器を縦続接続する





x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2進数減算

- $Z = X - Y$ → 加算で実現できる
  - Yの各ビットを反転（論理否定）する ($Y \rightarrow Y'$)
    - $Y + Y' = 2^N - 1$
      $-Y = Y' + 1 - 2^N$
  - $Z = X + Y' + 1 - 2^N$

Ex: 00101 + 11010 = 11111

-2^N の項はNビット表現では無視できる

```
                  1        1        1        1        1
  00111        00111    00111    00111    00111    00111
- 00101  ⇨  + 11010  + 11010  + 11010  + 11010  + 11010
ビット反転        10       11       10       10       10
                   0       10      010     0010    00010
                                                   result
```

00111	− 00101	= 00010
7	− 5	= 2

このビットは無視

2進数減算

- 減算結果が負の場合は、2進数表現となる

```
                    1        1        0        0        0
  00101         00101    00101    00101    00101    00101
- 00111  ⇒    + 11000  + 11000  + 11000  + 11000  + 11000
ビット反転        10       01       01       01       00
                  0        10       110      1110     11110
                                                      result
```

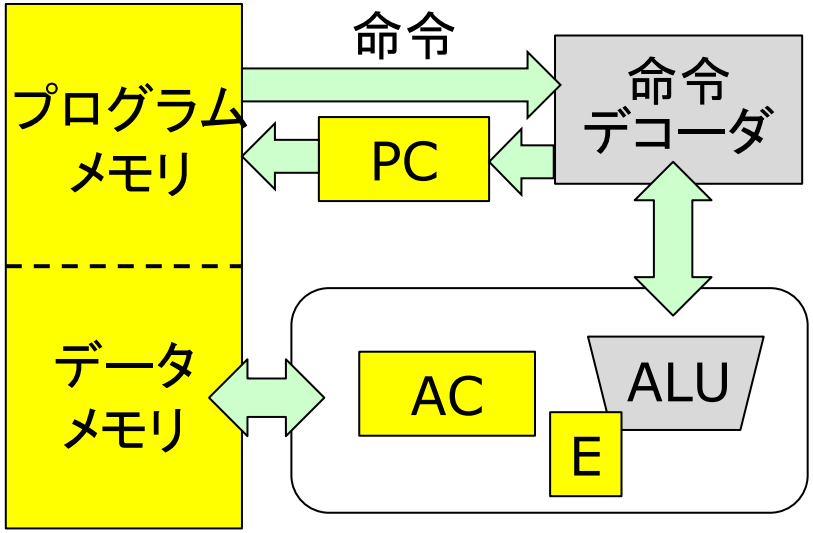
00101	- 00111	= 11110
5	- 7	= -2

このビットは無視

本講義で取り上げる16ビットマイクロプロセッサ

[コンピュータアーキテクチャ, M.Morris Mano著, 1999]

- メモリ : アドレス(12 bits)、データ(16 bits) → 4096 words
- アキュムレータ (AC) : データ一時格納
- プログラムカウンタ (PC) : 実行命令の格納場所
- 命令デコーダ : 命令種別の判定
  - データ転送、演算
  - プログラム制御
- ALU（算術論理演算器）
  - 四則演算（+のみ）
  - 論理演算（&のみ）
  - シフト演算（<<, >>）
- 状態レジスタ (E)
  - 加算桁上げ、シフトあふれ



本講義で取り上げる計算機では
極めて限られた演算しかできない

プログラミング言語の抽象度

■ **機械語プログラム：**

- 計算機が認識できる2値データによる命令表現

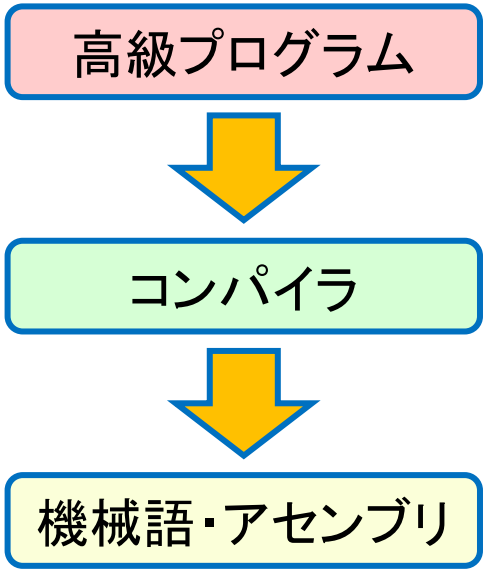
■ **アセンブリ言語プログラム：**

- テキスト形式の命令表現 → 機械語命令と等価

■ **高級プログラミング言語：**

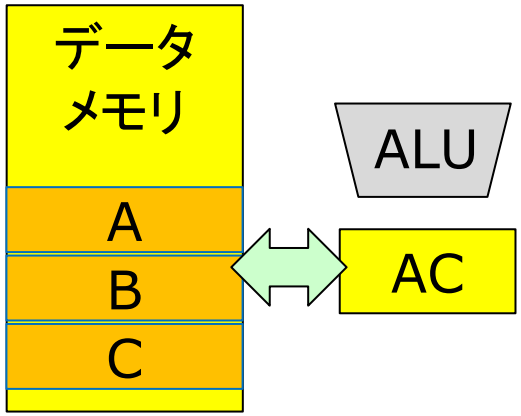
- 大規模ソフトウェア記述用（C/C++, Java, Fortran等）

→ 本講義では、アセンブリプログラム・機械語プログラムを扱う



簡単な計算機プログラム例 (1)

LDA A	: $AC \leftarrow M[A]$
ADD B	: $AC \leftarrow AC + M[B]$
STA C	: $M[C] \leftarrow AC$



■ アセンブリ命令 : [命令名] [ラベル名]

- LDA = ロード命令、ADD = 加算命令、STA = ストア命令
- ラベル名 (A, B, C) : メモリ番地（アドレス値でなく記号で表記）

■ レジスタ転送記述

- HW構成要素（レジスタ、メモリ）の間のデータ転送と演算を定義
- AC（アキュムレータ）
- M[A]：A番地のメモリデータ

高級プログラミング言語と アセンブリ言語の違い

```
c = a + b;
```

LDA A	: $AC \leftarrow M[A]$
ADD B	: $AC \leftarrow AC + M[B]$
STA C	: $M[C] \leftarrow AC$

■ 高級プログラミング言語

- 命令セットに非依存 → 異種CPU/命令セットで共通化可能
- 記述の抽象化により、より複雑な処理が簡素に表現できる
- 多様なデータ型のプログラミング変数が幾らでも使える

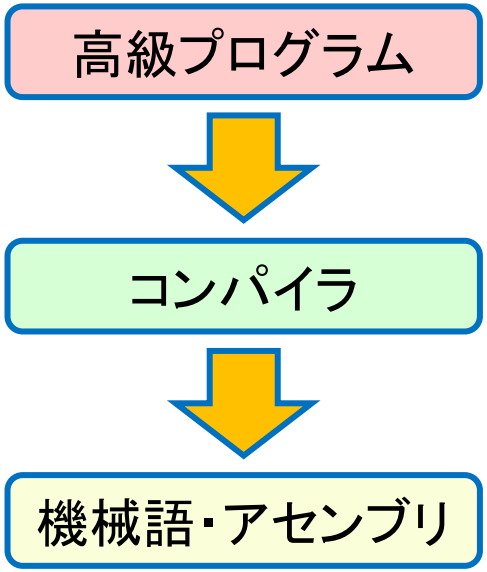
■ アセンブリ言語

- 計算機ハードウェアが直接解釈する機械語と等価
- 「プログラミング変数」は、レジスタとメモリ番地のみ
- 実行できる演算種類も限られる

命令セット・機械語・アセンブリ言語の重要性

- **プログラムの「良し悪し」：処理速度**
  - 高級プログラムがどのような機械語プログラムにコンパイラが変換するのかを知ることで、より効率的なコードを書くことが出来る
- **命令セット：SWとHWの接点**
  - SW：実行可能形式（実行バイナリ）
  - HW：論理回路設計「仕様」
- **命令セットを起点としたHW設計手法の習得が本講義の大きな目的**

  →そのために、命令セット自体は単純なものを取り上げる
- **命令セットは非常に寿命が長い**
  - インテルx86命令セット：1978年～現在



Intel マイクロプロセッサの集積度

年	型名	ワード長	クロック周波数	トランジスタ数
1978	8086	16-bits	～ 10MHz	29,000
1982	80286	16-bits	～ 25MHz	134,000
1985	80386	32-bits	～ 40MHz	275,000
1989	80486	32-bits	～ 150MHz	1.2M
1993	Pentium	32-bits	～ 233MHz	3.1M ～ 4.5M
1995	Pentium-Pro	32-bits	～ 200MHz	5.5M
1997	Pentium II	32-bits	～ 450MHz	7.5M
1999	Pentium III	32-bits	0.4G ～ 1.4GHz	9.5M ～ 21M
2000	Pentium 4	32/64-bits	1.3G ～ 3.8GHz	42M ～ 184M
2003	Pentium M	32-bits	0.9G ～ 2.6GHz	140M
2006	Core 2	64-bits	1.0G ～ 3.3GHz	169M ～ 411M
2008	Core i7	64-bits	～ 3.2GHz	731M

M : 10^6
G : 10^9

クロック周波数は10年以上停滞

インテルx86命令セットの形式

1）機械語

C7 45 44 08 00 00 00

→ 命令バイト系列

decoding sequence ↓

C7	“MOV” 命令
45	“modR/M” byte
44	8-bit アドレス オフセット値
08 00 00 00	32-bit 即値

1）アセンブリ言語

mov dword ptr [rbp+44h],8

1）レジスタ転送記述

M_{32}[rbp+44h] ← 00000008h

減算プログラム（減算命令がない場合）

LDA B	: $AC \leftarrow M[B]$
CMA	: $AC \leftarrow \overline{AC}$ （AC をビット反転）
INC	: $AC \leftarrow AC + 1$
ADD A	: $AC \leftarrow AC + M[A]$
STA C	: $M[C] \leftarrow AC$

```
  00111          1
- 00101 ⇨     00111
                + 11010
                 00010
```

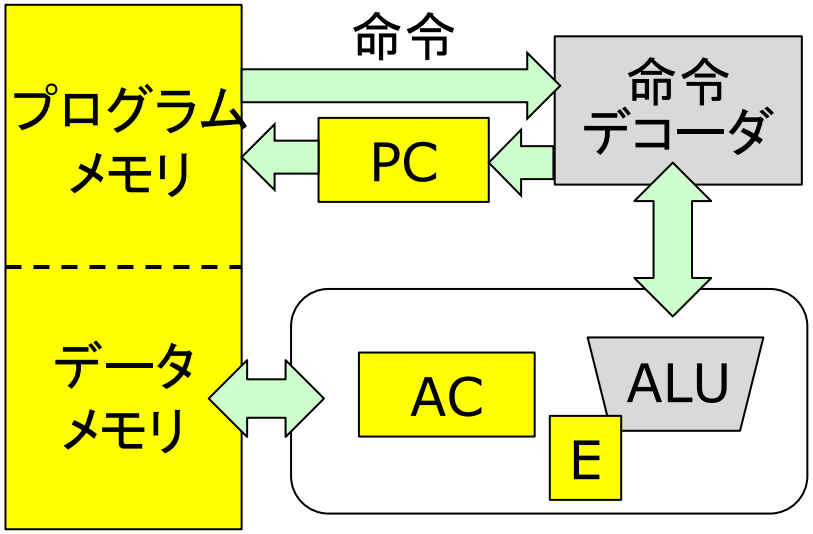
ビット反転

```
00111 - 00101 = 00010
  7   -   5   =   2
```

全ての演算を行う命令が揃っていなくとも、命令を組合せることで機能を補うことができる

16ビットマイクロプロセッサ命令セット

- 命令オペランド : 命令が参照・代入するレジスタやメモリ
  - メモリ : アドレス(12 bits)、データ(16 bits)
  - アキュムレータ (AC)
  - プログラムカウンタ (PC)
  - 状態レジスタ (E)
- 命令形式:
  - メモリ参照命令
    - 直接アドレス
    - 間接アドレス
  - レジスタ参照命令



ADD A	: $AC \leftarrow AC + M[A]$	（直接アドレス）
ADD B I	: $AC \leftarrow AC + M[M[B]]$	（間接アドレス）

メモリ参照命令

命令	レジスタ転送記述	説明
AND adi	$AC \leftarrow AC \& M[ad]$	ACとメモリオペランドのAND演算
ADD adi	$AC \leftarrow AC + M[ad]$, $E \leftarrow C_{out}(16)$	ACとメモリオペランドの加算 （桁上げ$C_{out}(16)$をEに保存）
LDA adi	$AC \leftarrow M[ad]$	ACにメモリオペランドをロード
STA adi	$M[ad] \leftarrow AC$	ACをメモリにストア
BUN adi	$PC \leftarrow ad$	無条件にadに分岐
BSA adi	$M[ad] \leftarrow PC, PC \leftarrow ad + 1$	復帰アドレスをメモリにストア（退避）し、ad+1に分岐
ISZ adi	$M[ad] \leftarrow M[ad] + 1$, $IF(M[ad] + 1 = 0)\ THEN$ $PC \leftarrow PC + 1$	メモリオペランドをインクリメントし、その結果が0ならば次命令をスキップ

adi : 命令中で指定されるアドレス
ad : メモリオペランドのアドレス

直接アドレス : ad = adi
間接アドレス : ad = M[adi]

コンマで区切られた記述は「同時」に実行されるとする

レジスタ参照命令

命令	レジスタ転送記述	説明
CLA	$AC \leftarrow 0$	ACをクリア
CLE	$E \leftarrow 0$	Eをクリア
CMA	$AC \leftarrow \overline{AC}$	ACをビット反転
CME	$E \leftarrow \overline{E}$	Eをビット反転
CIR	$AC(14:0) \leftarrow AC(15:1), AC(15) \leftarrow E, E \leftarrow AC(0)$	ACとEを右回転シフト
CIL	$AC(15:1) \leftarrow AC(14:0), AC(0) \leftarrow E, E \leftarrow AC(15)$	ACとEを左回転シフト
INC	$AC \leftarrow AC + 1$	ACをインクリメント
SPA	$IF(AC(15) = 0)\ THEN\ PC \leftarrow PC + 1$	AC >= 0ならば次命令スキップ
SNA	$IF(AC(15) = 1)\ THEN\ PC \leftarrow PC + 1$	
SZA	$IF(AC = 0)\ THEN\ PC \leftarrow PC + 1$	
SZE	$IF(E = 0)\ THEN\ PC \leftarrow PC + 1$	
HLT	HALT	計算停止