計算機論理設計（O）講義ノート（担当：一色）

第 1 部

本講義の目的：

1．計算機アーキテクチャ全体（ハードウエア・ソフトウエアアーキテクチャ）の理解

- ソフトウエアアーキテクチャ：命令セット，アセンブリプログラミング
  - ➢ 限られた命令セットを組み合わせることで複雑な処理を実現するというソフトウエア本来の構成法や方法論を習得
- ハードウエアアーキテクチャ：計算機構成，プログラム実行動作，演算処理原理

2．計算機ハードウエアの各機能ブロックから全体アーキテクチャまでの全設計法の習得

- 論理回路による算術演算・論理演算・シフト演算回路の実現
- レジスタ転送記述法による計算機動作の表現
- 計算機制御回路の実現方法：
  - ➢ 組合せ回路（結線式：hardwired）
  - ➢ マイクロプログラム制御方式

1．計算機の用途：

① データ演算処理（電卓）

② 情報通信（情報端末，携帯電話，サーバー，ネットワーク機器）

③ 機械・装置・設備の電子制御→すべての家電機器も計算機を含む→「組込みシステム」

④ 汎用（多目的）：ＰＣ

2．情報の扱い：

① データ（数値）表現（2 進数）→整数型，固定小数点型，浮動小数点型

② 数値以外の情報表現（2 値変数の組：シンボル）→文字データ，

③ 情報処理・データ処理：論理回路（組合せ回路・順序回路）→計算機構成要素

3．計算機アーキテクチャ：

① ハードウエアアーキテクチャ：計算機構成

② ソフトウエアアーキテクチャ：

- ➢ 命令セット：計算機の基本動作を定義した手続き（命令）の集合
- ➢ システムソフトウエア：計算機実行を補助する（オペレーティングシステム，コンパイラ，ミドルウエア，デバイスドライバ）
- ➢ アプリケーションソフトウエア：用途別に準備した情報処理プログラム

4．計算機の構成要素：

① メモリ：アドレス（m ビット），データ（n ビット）→$2^m \times n$ ビット語（ワード）：例：m = 12，n = 16 →$2^{12} \times 16$ = 4096×16 ビット

- ➢ アドレスによって任意のメモリ番地の 1 ワードにアクセス（Random Access）
- ➢ RAM (Random Access Memory)：読み書き可能なメモリ→データ入力（書込み用），データ出力（読出し用），read/write 制御線（読出し・書込み指定）
- ➢ ROM(Read Only Memory)：読出し専用メモリ（アドレス・データのみ）→計算機動作中に書換えが必要ないデータを格納（プログラム，固定データ等）

➔ クロック入力のある同期式メモリが最近主流（本講義ではクロック入力のない非同期式を

扱う）

② レジスタ（情報の一時記憶）：データ入力・データ出力（n ビット），クロック入力→フリップフロップで実現

③ 演算回路：情報処理・データ処理を実行する組合せ回路

- 算術演算回路：四則演算（またはその一部）
- 論理演算回路：ワードの各ビットで論理演算を実行
- シフト演算回路：データの左シフト（×２）・右シフト（÷2）

④ 制御回路：メモリ・レジスタ・演算回路をプログラム命令列に従って制御

5．簡単な計算機構成（講義で扱うものを簡略化）：m = 12 ビットアドレス，n = 16 ビットデータ

① メモリ：プログラムとデータを保存（アドレス：m ビット，データ：n ビット）

② PC（プログラムカウンタ）：(m ビットレジスタ)

③ AC（アキュムレータ）：(演算結果データの一時保存用)（n ビットレジスタ）

④ ALU（Arithmetic Logic Unit：算術論理演算器）：命令セットの算術演算，論理演算を実行する組合せ回路（n ビット幅）

⑤ E（状態レジスタ）：加算の桁上げやシフトあふれを格納（1 ビットレジスタ）

6．計算機プログラム：目的の情報処理を実行する命令列

① 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

② アセンブリプログラム：テキスト形式の命令表現→直接機械語プログラムと対応

③ 高級プログラミング言語：大規模ソフトウエア記述用(C/C++，Java，Fortran)

→ 本講義では，主にアセンブリプログラムと機械語プログラムを扱う

7．2 進数整数表現：

① 符号なし整数（非負数）：$X = X(n-1:0) = \sum_{k=0}^{n-1} X(k)\cdot 2^k$ （$0 \le X \le 2^n - 1$）

② 符号付き 2 の補数：$X(n-1)$：最上位ビット→符号（1：負，0：正）

$$X = -X(n-1)\cdot 2^{n-1} + \sum_{k=0}^{n-2} X(k)\cdot 2^k = -X(n-1)\cdot 2^{n-1} + X(n-2:0)$$

- $-2^{n-1} \le X \le 2^{n-1} - 1$
- $-X = \overline{X} + 1$（$\overline{X}$：X のビット反転）

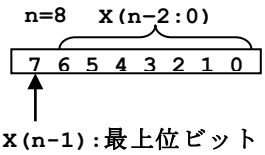
$$\overline{X} = -\overline{X(n-1)}\cdot 2^{n-1} + \overline{X(n-2:0)}$$

$$X + \overline{X} = -\left(X(n-1) + \overline{X(n-1)}\right)\cdot 2^{n-1} + X(n-2:0) + \overline{X(n-2:0)}$$

$$X(n-1) + \overline{X(n-1)} = 1, \quad X(n-2:0) + \overline{X(n-2:0)} = 2^{n-1} - 1$$

$$\therefore X + \overline{X} = -2^{n-1} + 2^{n-1} - 1 = -1$$

$$\therefore -X = \overline{X} + 1$$




```
n = 4:
0111 (7)
0110 (6)
0101 (5)
0100 (4)
0011 (3)
0010 (2)
0001 (1)
0000 (0)
1111 (-1=-8+7)
1110 (-2=-8+6)
1101 (-3=-8+5)
1100 (-4=-8+4)
1011 (-5=-8+3)
1010 (-6=-8+2)
1001 (-7=-8+1)
1000 (-8=-8+0)
```

例１：加算プログラム：C = A + B

LDA	A	：	$AC \leftarrow M[A]$	（メモリのＡ番地のデータを AC にロード）
ADD	B	：	$AC \leftarrow AC + M[B]$	（メモリのＢ番地と AC を加算，AC に保存）
STA	C	：	$M[C] \leftarrow AC$	（AC をメモリのＣ番地に書込み）

```
  00000110 (carry)
  00000111 (7)
+ 00000101 (5)
  00001100 (12)
```

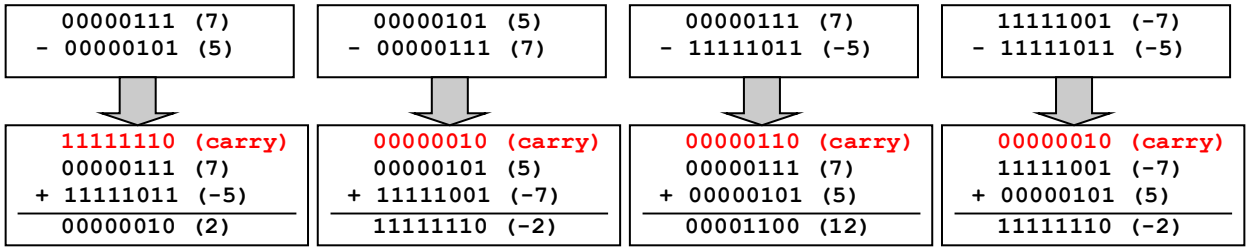
```
  00000010 (carry)
  11111001 (-7)
+ 00000101 (5)
  11111110 (-2)
```

```
  11111110 (carry)
  00000111 (7)
+ 11111011 (-5)
  00000010 (2)
```

```
  11110010 (carry)
  11111001 (-7)
+ 11111011 (-5)
  11110010 (-12)
```

例 2：減算プログラム：C＝A－B（減算命令がない場合）

LDA	B	：	$AC \leftarrow M[B]$	（メモリのＢ番地のデータを AC にロード）
CMA		：	$AC \leftarrow \overline{AC}$	（AC をビット反転）
INC		：	$AC \leftarrow AC + 1$	（AC をインクリメント）
ADD	A	：	$AC \leftarrow AC + M[A]$	（AC とメモリのＡ番地を加算）
STA	C	：	$M[C] \leftarrow AC$	（AC をメモリのＣ番地に書込み）



８．命令セットの構成

① 命令の機能（1 つの命令が 2 つ以上の機能を持つ場合もある）

- メモリ参照（例：LDA，STA，ADD）→上の例は直接アドレスを示す
- 演算：算術・論理・シフト（例：ADD，CMA，INC）
- プログラム実行制御：無条件分岐，条件分岐，関数呼出し・復帰（call/return）
- 入出力処理：外部装置との通信

② オペランド（演算対象データ）

- メモリオペランド：メモリアドレスを命令で指定（直接アドレス・間接アドレス）
- レジスタオペランド：対象となるレジスタが 2 つ以上ある場合はレジスタ番号を命令で指定→例では AC 一つなので指定の必要なし
- 即値オペランド：定数オペランドを命令で指定（今回の命令セットにはない）

９．簡単な命令セットの例

＊ コンマで区切られた記述は「同時」に実行されるとする→補足説明（波形図・クロック）
＊ $Cout(16)$ ：16 ビット加算器の桁上げ
＊ ad ：メモリオペランドのアドレス→ $M[ad]$ ：アドレス値 ad におけるメモリの値
＊ adi ：命令中で指定されるアドレス値
  - 直接アドレス： $ad = adi$ （adi がメモリオペランドのアドレス値）
  - 間接アドレス： $ad = M[adi]$ （adi に格納されているメモリの値がメモリオペランドのアドレス値）
＊ 実行中の命令のアドレスは PC － 1 とする（次の命令のアドレスは PC とする）→つまり PC は命令実行の前にインクリメントされていると想定する．
＊ $AC(15)$ ： AC の 15 ビット目， $AC(14:0)$ ： AC の 0 ビット目から 14 ビット目

命令	RT（Register Transfer）記述	
AND	$AC \leftarrow AC \,\&\, M[ad]$	**AC とメモリオペランドの AND**
ADD	$AC \leftarrow AC + M[ad], E \leftarrow Cout(16)$	**AC とメモリオペランドの加算（桁上げを E に格納）**
LDA	$AC \leftarrow M[ad]$	**AC にメモリオペランドをロード**
STA	$M[ad] \leftarrow AC$	**メモリに AC をストア**
BUN	$PC \leftarrow ad$	**無条件に ad に分岐**
BSA	$M[ad] \leftarrow PC, PC \leftarrow ad + 1$	**復帰先命令アドレスをストア（退避）し，ad+1 に分岐**

ISZ	$M[ad] \leftarrow M[ad]+1,$ $IF(M[ad]+1=0)\ THEN\ PC \leftarrow PC+1$	メモリオペランドをインクリメント,その結果が 0 ならば次の命令をスキップ
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** の回転右シフト
CIL	$AC(15:1) \leftarrow AC(14:0), AC(0) \leftarrow E, E \leftarrow AC(15)$	**AC** と **E** の回転左シフト
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** ならば次の命令をスキップ
SNA	$IF(AC(15)=1)\ THEN\ PC \leftarrow PC+1$	**AC < 0** ならば次の命令をスキップ
SZA	$IF(AC=0)\ THEN\ PC \leftarrow PC+1$	**AC = 0** ならば次の命令をスキップ
SZE	$IF(E=0)\ THEN\ PC \leftarrow PC+1$	**E = 0** ならば次の命令をスキップ
HLT	$HALT$	計算機を停止
その他：入出力命令		

① メモリアドレス参照命令：ADD, AND, LDA, STA, ISZ

② 算術論理演算命令：ADD, AND, CLA, CLE, CMA, CME, INC

③ シフト命令：CIR, CIL

④ プログラム実行制御命令：BUN, BSA, ISZ, SPA, SNA, SZA, SZE, HLT

⑤ 状態レジスタ E を変更する命令：ADD, CLE, CME, CIR, CIL

→ 必要最小限の算術論理命令（加算，論理積，ビット反転）

→ 必要最小限のプログラム実行制御（分岐，関数呼出し）

→ 演算に使用できるレジスタはただ１つだけ

→ プログラム変数は基本的にすべてメモリに格納（すべてグローバル変数）

→ サブルーチンの再帰呼出しができない

本講義で取り上げる基本計算機アーキテクチャの特徴（意図）

- 簡単な構成と命令セットを組み合わせて複雑な処理を実現する方法論・設計論を学ぶ教材
- 計算機ハードウエアの全体設計を講義期間内で習得できるよう，設計対象となるハードウエア構成要素を極力簡単化
- 計算機構成の歴史的背景：
  - 計算機は半導体集積回路が登場する前から存在（バキュームチューブ，リレー，真空管，トランジスタ素子）
  - 集積回路以前はハードウエア資源が非常に高価
  - 回路規模・プログラムサイズに大きな制約
  - 計算機アーキテクチャ（ハード・ソフト）でなされた様々な工夫を学ぶ

→ 現在の計算機アーキテクチャ（パイプライン，スーパースカラー，マルチスレッド）がどのような技術的背景で生まれてきたかを学ぶことも重要

１０．　シフト演算：

① 論理シフト：データを「符号なし」と見なす

- 論理右シフト（×2）：$AC(14:0) \leftarrow AC(15:1), AC(15) \leftarrow 0$
- 論理左シフト（÷2）：$AC(15:1) \leftarrow AC(14:0), AC(0) \leftarrow 0$

② 算術シフト：データを「2 の補数」と見なす

- 算術右シフト（×2）：$AC(14:0) \leftarrow AC(15:1), AC(15) \leftarrow AC(14)$（符号拡張）
- 算術左シフト（÷2）：$AC(15:1) \leftarrow AC(14:0), AC(0) \leftarrow 0$（論理左シフトと同じ）

```
1111:(n=4)
0111(論理右シフト:15/2=7.5→7)
1111(算術右シフト:-1/2=-0.5→-1)
1110(論理左シフト:15*2=30=16+14(オーバーフロー)
1110(算術左シフト：-1*2=-2
```

```
0111:(n=4)
0011(論理右シフト:7/2=3.5→3)
0011(算術右シフト:7/2=3.5→3)
1110(論理左シフト:7*2=14
1110(算術左シフト：7*2=14→-8+6（オーバーフロー）
```

③ 回転シフト：データのビット列をシフト方向に回転する

- 回転右シフト：$AC(14:0) \leftarrow AC(15:1), AC(15) \leftarrow AC(0)$
- 回転左シフト：$AC(15:1) \leftarrow AC(14:0), AC(0) \leftarrow AC(15)$

各シフト演算を実現する命令列（元データは AC にあり，E は 0 とは限らない）

論理右シフト	論理左シフト	算術右シフト	回転右シフト	回転左シフト
CLE (E←0) CIR	CLE (E←0) CIL	CLE (E←0) SPA (skip if AC≥0) CME (E← E'=1) CIR	CLE (E←0) CIR SZE (skip if E=0) AND Y (M[Y]=1 とする)	CLE (E←0) CIL SZE (skip if E=0) INC

④ 倍精度データのシフト：n ビットデータ幅の計算機において 2n ビットデータのシフト演算を実現（メモリ上の AL，AH アドレスの倍精度データをシフト）

論理右シフト	算術右シフト	論理左シフト
CLE　(E ← 0) LDA　AH CIR　(E ← AC(0)) STA　AH LDA　AL CIR　(AC(15) ← E) STA　AL	CLE　(E ← 0) LDA　AH SPA CME CIR　(E ← AC(0)) STA　AH LDA　AL CIR　(AC(15) ← E) STA　AL	CLE　(E ← 0) LDA　AL CIL　(E ← AC(15)) STA　AL LDA　AH CIL　(AC(0) ← E) STA　AH

１１．　加減算

①　加算：ADD 命令

②　減算(A － B)：CMA, INC によって減数(B)を 2 の補数に変換し非減算数(A)を足す

③　絶対値計算：　C = |A|

	LDA	A	： $AC \leftarrow M[A]$	（メモリのＡ番地のデータを AC にロード）
	SNA		： $IF(AC(15)=1)\ THEN\ PC \leftarrow PC+1$	（**AC ＜ 0 ならば次の命令をスキップ**）
	BUN	L0	： $PC \leftarrow L0$	（命令アドレス L0 に無条件分岐）
	CMP		： $AC \leftarrow \overline{AC}$	（AC をビット反転）
	INC		： $AC \leftarrow AC+1$	（AC をインクリメント）
L0,	STA	C	： $M[C] \leftarrow AC$	（AC をメモリのＣ番地に書込み）
	HLT			

④　倍精度加算：32 ビットデータを AH, AL と BH, BL で表し，その和を CH, CL に格納する．

{CH, CL} = {AH, AL} + {BH, BL}　（倍精度加算）（P206 バグあり）

LDA	AL	： $AC \leftarrow M[AL]$
ADD	BL	： $AC \leftarrow AC+M[BL], E \leftarrow C_{out}(16)$
STA	CL	： $M[CL] \leftarrow AC$
CLA		： $AC \leftarrow 0$
CIL		： $AC(0) \leftarrow E$ （AC の最下位ビット以外は変化なし）
ADD	AH	： $AC \leftarrow AC+M[AH]$
ADD	BH	： $AC \leftarrow AC+M[BH]$
STA	CH	： $M[CH] \leftarrow AC$
HLT		

１２．　乗算(P = A * B) : 筆算方式を応用（A, B： n/2 ビット，P： n ビット，すべて符号なし非負数）

① $P_k \leftarrow A \cdot 2^k \cdot B(k)$ → （A の k ビット左シフト）＊（B の k ビット目）

② $P \leftarrow \sum_{k=0}^{n/2-1} P_k$

n=8 ビットで説明（実際は n=16 ビット）

乗算：M[P] = M[A] * M[B]　（積は符号なし非負数とする）

M[A]	M[B]	P_k	M[P]
00001011	00001101	00001011	00001011
00010110	00000110	00000000	00001011
00101100	00000011	00101100	00110111
01011000	00000001	01011000	10001111

```
     00001011  (11)
x    00001101  (13)
-----------------------
     00001011  (P0)
     00000000  (P1)
-----------------------
     00001011  (P)
     00101100  (P2)
-----------------------
     00110111  (P)
     01011000  (P3)
-----------------------
     10001111 (P=143)
```

```
     11111011  (-5)
x    00001101  (13)
-----------------------
     11111011  (P0)
     00000000  (P1)
-----------------------
     11111011  (P)
     11101100  (P2)
-----------------------
     11100111  (P)
     11011000  (P3)
-----------------------
     10111111 (P=-65)
```

被乗数Ａが２の補数表現（$-2^{n-2} \leq A < 2^{n-2} -1$）の場合は，積Ｐも２の補数表現として同じアルゴリズムで計算できる

```
L0, CLE
    LDA B
    CIR
    STA B
    SZE
    BUN L1
    BUN L2
L1, LDA A
    ADD P
    STA P
    CLE
L2, LDA A
    CIL
    STA A
    ISZ C
    BUN L0
    HLT
C, DEC -8
P, DEC 0
```

M[B] ← M[B] >> 1
E ← B(k)

B(k) = 1:
P_k ← M[A]
M[P] ←M[P]+P_k

M[A] = M[A] << 1
M[C] = M[C] + 1

（A, B： n/2 ビット，P： n ビット，すべて２の補数）

$$n = 8, n' = n/2 = 4:$$

$$B = -2^{n'-1} \cdot B(n'-1) + B(n'-2:0)$$

$$P = A \times B = A \cdot \left(-2^{n'-1} \cdot B(n'-1) + B(n'-2:0)\right)$$

$$= -A \cdot B(n'-1) \cdot 2^{n'-1} + A \cdot B(n'-2:0)$$

$$= -P_{n'-1} + \sum_{k=0}^{n'-2} P_k$$

$$P_S = -P_{n'-1}, P_M = \sum_{k=0}^{n'-2} P_k$$

4x4=8 ビットで説明 $(P_S = -A \cdot B(3) \cdot 2^3, P_M = A \cdot B(2:0))$
（実際は 8x8=16 ビット）

```
     00001011  (11)
x    11111011  (-5)
-----------------------
     00001011  (P0)
     00010110  (P1)
-----------------------
     00100001  (P)
     00000000  (P2)
-----------------------
     00100001  (PM=33)
     10101000  (PS=-11*2^3)
-----------------------
     11001001  (P=-55)
```

```
L0, CLE
    LDA B
    CIR
    STA B
    SZE
    BUN L1
    BUN L4
L1, LDA C
    INC
    SZA
    BUN L2
    LDA A
    BUN L3
L2, LDA A
    CMA
    INC
L3, ADD P
    STA P
    CLE
L4, LDA A
    CIL
    STA A
    ISZ C
    BUN L0
    HLT
C, DEC -8
P, DEC 0
```

M[B] ← M[B] >> 1
E ← B(k)

B(k) = 1:
C + 1 = 0: P_k ← – M[A]
C + 1 ≠ 0: P_k ← M[A]
M[P] ←M[P]+P_k

M[A] = M[A] << 1
M[C] = M[C] + 1

１３．　プログラムループと間接アドレス

```
        CLA              ： AC ← 0
L0,     ADD     PTR   I ： AC ← M[M[PTR]]      （PTR 番地の間接アドレス）
                          （adi = PTR, ad = M[PTR], AC ← M[ad]）
        ISZ     PTR      ： M[PTR] ← M[PTR]+1, IF(M[PTR]+1 = 0) THEN PC ← PC+1
                          （M[PTR]+1=0 ならば次の命令をスキップ）
                          （注：このスキップ条件は起こらない）
        ISZ     CTR      ： M[CTR] ← M[CTR]+1, IF(M[CTR]+1 = 0) THEN PC ← PC+1
                          （M[CTR]+1=0 ならば次の命令をスキップ）
        BUN     L0       ： PC ← L0        （命令アドレス L0 に無条件分岐）
        STA     SUM      ： AC ← ‾AC       （AC をビット反転）
        HLT
PTR     HEX     150
CTR     DEC     – 100
SUM     DEC     0
        ORG     150      ：次の行を 0x150 番地から開始
        DEC     75       ：0x150 番地の値
        DEC     62       ：0x151 番地の値
        …
        DEC     23       ：0x1B3 番地の値
        END
```

１４．　サブルーチン

```
        LDA     X
        BSA     SH4
        STA     X
        LDA     Y
        BSA     SH4
        STA     Y
        HLT
X,      HEX     1234
Y,      HEX     4321
SH4,    HEX     0        ：  ここに戻りアドレスを格納
        CIL
        CIL
        CIL
        CIL
        AND     MSK      ： AC ← AC & M[MSK]
        BUN     SH4   I ：サブルーチンから復帰（SH4 に格納されているアドレスへジャンプ）
MSK,    HEX     FFF0
        END
```

サブルーチンに２個以上の引数を渡す方法：

```
        LDA   X        ：第１引数をロード
        BSA   OR
        HEX   3AF6     ：第２引数の値
        STA   Y
        HLT
X,      HEX   7B95
Y,      HEX   0        ：結果格納（7B95 & 3AF6）


OR,     HEX   0        ：ここに戻りアドレスを格納
        CMA            ：
        STA   TMP      ：
        LDA   OR  I    ：                （OR 番地の間接アドレス）
        CMA            ：
        AND   TMP      ：
        CMA            ：
        ISZ   OR       ：
                       （M[OR]+1=0 ならば次の命令をスキップ）
                       （注：このスキップ条件は起こらない）
        BUN   OR  I    ：サブルーチンから復帰（OR に格納されているアドレスへジャンプ）
TMP,    HEX   0
        END
```