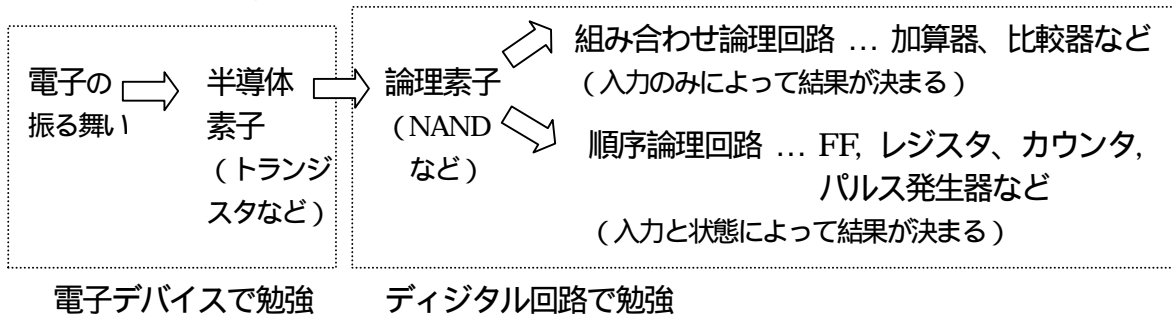


コンピュータアーキテクチャ

参考資料

1. この授業で勉強すること

1.1. 今までに勉強したこと

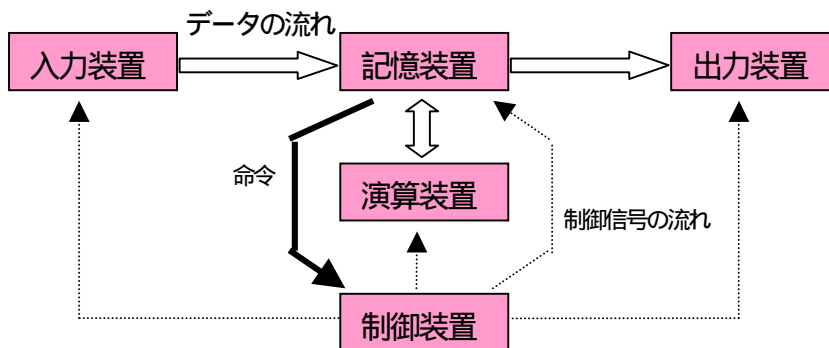


1.2. この授業で勉強すること

論理回路 ⇔ コンピュータ

- コンピュータの動作原理 (機械語)
- アセンブラ (アセンブリ言語)
- I/O (入出力) 装置の制御
- 各自、自由なテーマで周辺機器を用いたプログラム作成

2. コンピュータの基本構成



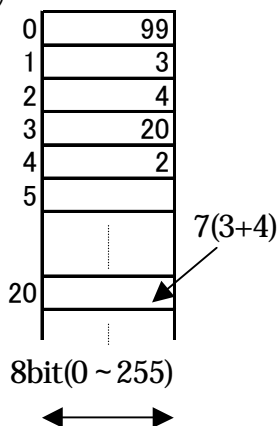
3. ストアドプログラム方式

「あらかじめ情報処理の方式や順序を数値データに変換し、記憶装置に記憶させ、この命令データに従って時系列的に処理装置を稼働させる方式」

【例】 3 + 4 [20 番地]

記憶装置 (メモリ)

アドレス
(番地)

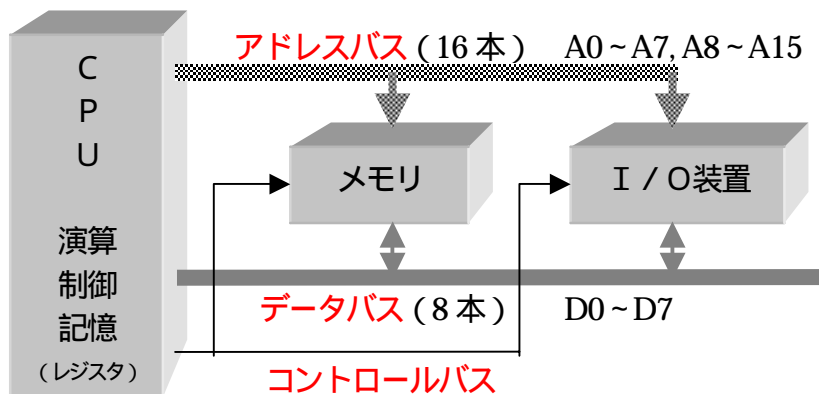


命令表 (0~255)

コード (機械語)	内容
99	(次の番地の内容) + (次の次の番地の内容)の結果を次の次の次に書いてある番地に格納
2	STOP

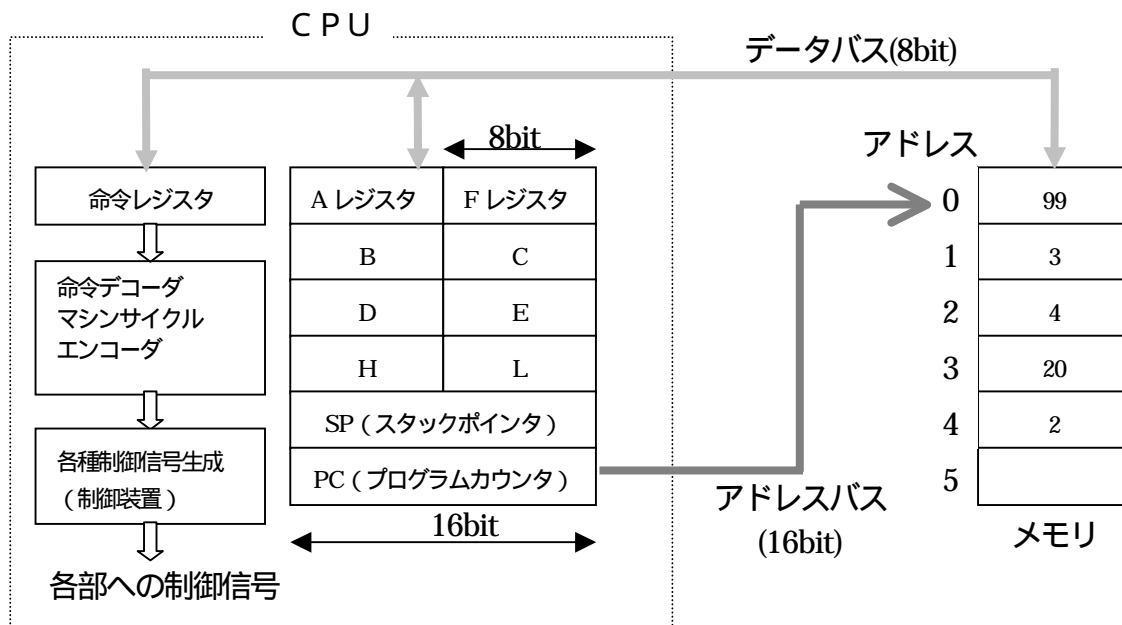
4. マイコンの構成

4.1. CPU 8085A



- 8bit...データバスが8本
 2^8 、0~255 の数を表現可能
- アドレスバス...16本、 $2^{16}=64K$ 、64K 個の番地を指定可能
- 構成...
 演算 ALU(算術論理装置)
 制御 タイミング制御部、割り込み制御部、シリアル I/O 制御部
 メモリの一部 レジスタ群

4.2. レジスタ群



(1) 汎用レジスタ

CPU 8085AにはAレジスタ、Bレジスタ、Cレジスタ、Dレジスタ、Eレジスタ、Hレジスタ、Lレジスタの7つの汎用レジスタ（多目的に利用できるレジスタ）が内蔵されています。これらの各レジスタはそれぞれ8bit（1バイト）の記憶容量があります。すなわち、各レジスタは8個のF.F.（フリップフロップ）を並べたものです。

Aレジスタは、アキュムレータとも呼ばれます。またBレジスタとCレジスタは、2つまとめて16bitのレジスタとして使用することもできます。BレジスタとCレジスタを2つまとめて16bitのレジスタとして使用する場合、BCペアレジスタと呼びます。同様にDレジスタとEレジスタをまとめて16bitのDEペアレジスタとして使用できます。HレジスタとLレジスタもまとめて16bitのHLペアレジスタとして使用できます。

(2) Fレジスタ

Fレジスタは、8bitのレジスタです。演算結果が正、零、負、桁上がりなどに対応して、各bitの値が自動的に変化します。例えば、演算結果が零であれば、あるビットが1になります。このビットを見れば、演算結果が零であるかどうかを1または0（旗をあげる／あげない）で知ることができます。このためFレジスタはフラグ（旗を意味する）レジスタと呼ばれます。

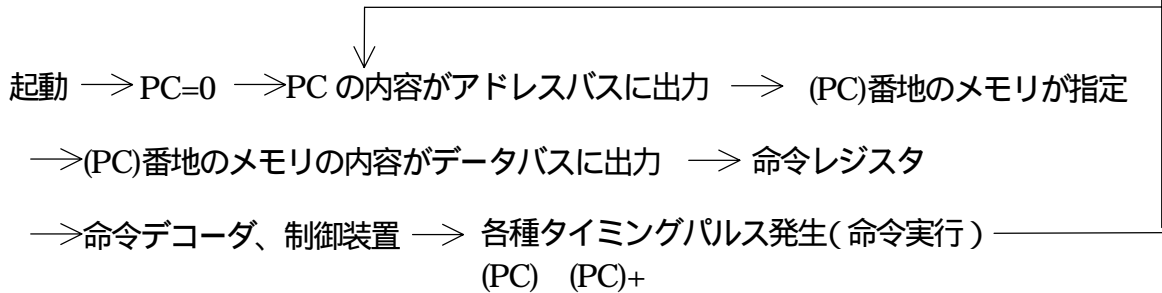
(3) 命令レジスタ

命令レジスタも8bitのレジスタで、メモリ上に記憶されている命令データ（機械語コード）を一時的に記憶するためのレジスタです。命令レジスタに記憶された機械語コードは命令デコーダなどで各部への制御信号に変換されます（上図参照）。

(4) PC（プログラムカウンタ）

プログラムカウンタ(PC)は、次に実行する命令が書かれているメモリの番地（アドレス）を格納するためのレジスタです。

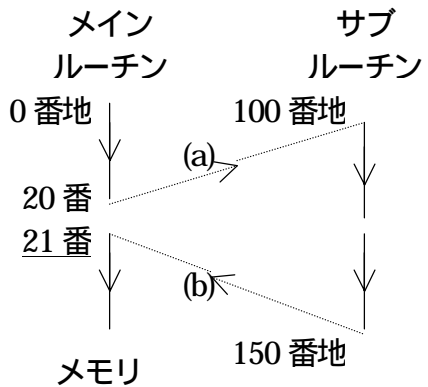
PC は起動時に 0 にリセットされるため、前ページの図のように、まず 0 がアドレスバスを通じてメモリに番地として与えられます。つぎに 0 番地のデータ(99)がデータバスを通じて命令レジスタに転送されます。この命令データは、命令デコーダ及び制御装置で各部への制御信号(タイミングパルス)が生成され命令が実行されます。ここで PC の値は、次の命令が書かれている番地になるまで(例では 4 つ)増えます。このような操作をプログラムの終了命令(STOP 命令)があるまで繰り返します。



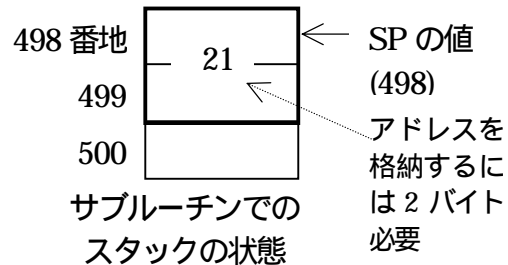
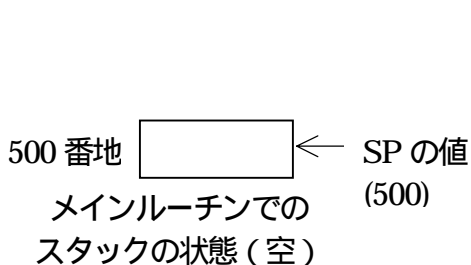
(5) SP (スタックポインタ)

スタックポインタ(SP)は、割り込み処理やサブルーチンを終了したときに、実行すべき番地が書かれている番地を格納するためのレジスタで、スタックの一番上の番地を記憶しています。

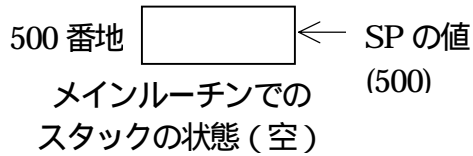
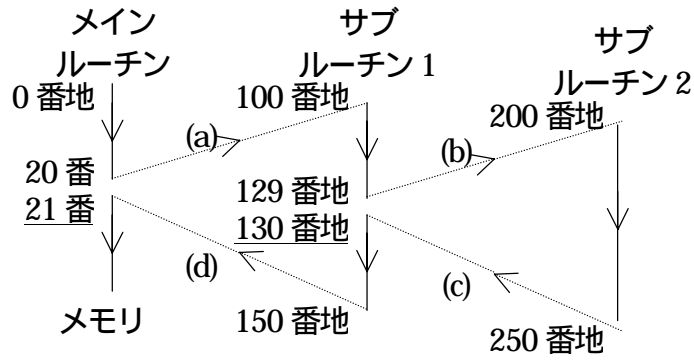
【例 1】 1 回のサブルーチンコール (SP の初期値が 500 の場合)



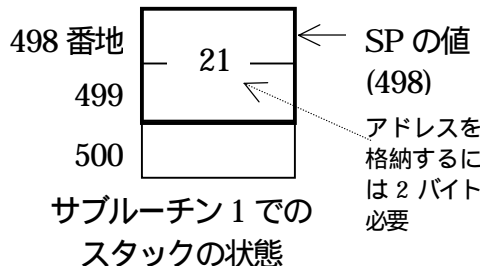
- (a) サブルーチンコール時
 $[(SP - 1)], [(SP - 2)]$ 21
 (SP) (SP) - 2
 (PC) 100
- (b) リターン時
 (PC) $[(SP + 1)], [(SP)]$
 (SP) (SP) + 2



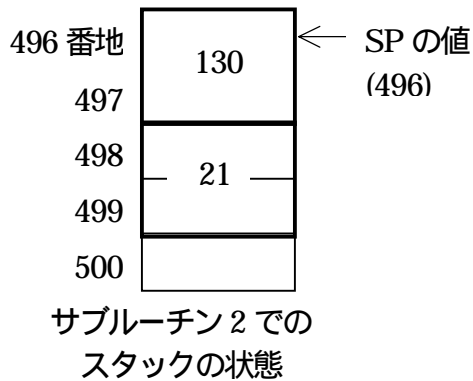
【例 2】 2重のサブルーチンコール (SP の初期値が 500 の場合)



- (a) サブルーチン 1 コール時
 [(SP - 1)], [(SP - 2)] 21
 (SP) (SP) - 2
 (PC) 100

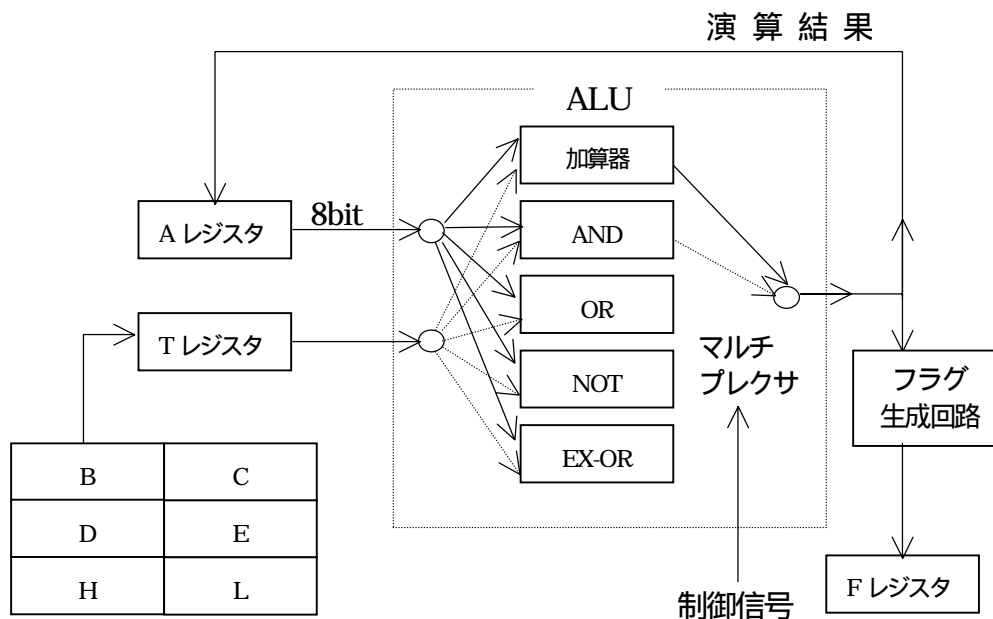


- (b) サブルーチン 2 コール時
 [(SP - 1)], [(SP - 2)] 130
 (SP) (SP) - 2
 (PC) 200



- (c) サブルーチン 1 にリターン時
 (PC) [(SP+1)], [(SP)]
 (SP) (SP)+2
- (d) メインルーチンにリターン時
 (PC) [(SP+1)], [(SP)]
 (SP) (SP)+2

4.3. 演算部



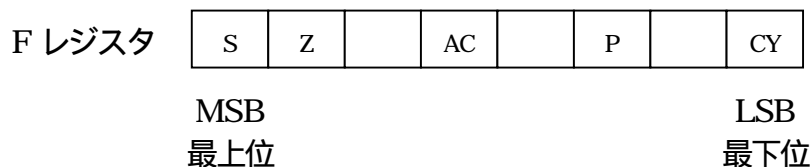
CPU 8085A の場合、T レジスタ (一時レジスタ) に B, C, D, E, H, L などのレジスタの内容が転送され、演算は A レジスタと T レジスタとの間で行われます。A レジスタと T レジスタの内容は ALU (Arithmetic and Logical Unit, 演算論理ユニット) に入力され、ALU 内で演算が実行されます。ALU には加算器、AND 回路、OR 回路、NOT 回路、EX-OR 回路がそれぞれ 8bit 分内蔵されています。どの演算を実行するかは、命令データから生成された制御信号を ALU 内のマルチプレクサに与えることによって選択されます。

演算結果は A レジスタに格納されます。例えば A レジスタの初期値が 0、B レジスタの初期値が 2 の場合、A+B の加算命令を実行する度に A レジスタの内容は 2, 4, 6, ... と結果が累算されていくため、A レジスタはアキュムレータ (accumulator, 累算器) とも呼ばれています。また減算なども実行できるように、A レジスタには補数機能、左右シフト機能などが備えられています。

一方、演算結果はフラグを生成する論理回路にも入力され、F レジスタに演算結果に対応したフラグがセットされます。

4.4. 条件分岐

(1) フラグ



Z=1 ... 演算結果が 0	Z=0 ... 演算結果が 0 でない
CY=1 ... 桁上げあり	CY=0 ... 桁上げなし
S=1 ... MSB が 1 (2 の補数では負)	S=0 ... MSB が 0 (2 の補数では正)
P=1 ... 1 が偶数個	P=0 ... 1 が奇数個
AC=1 ... 4bit 目から 5bit 目に桁上げあり	

(2) 条件判定

【例1】 if(A=B) goto L1; (もしA=BならL1番地にジャンプしたいとき)

(a) A-Bを計算 フラグが変化 (A=Bなら演算結果が0になるため、Z=1になる)

(b) Z=1なら goto L1

PC L1

【例2】 if(条件) goto L1; (もし条件を満たせばL1番地にジャンプしたいとき)

(a) A-Bを計算 フラグが以下のように変化

	条件	Z	CY	S
	A = B	1		
符号なし (0~255)	A = B		0	
	A < B		1	
符号あり (-128~127)	A = B			0
	A < B			1

(b) 以上のフラグを用いて条件分岐を行う。たとえば、if(A<B)goto L1;の場合でA,Bの値を符号なし(unsigned)のときには、CY=1ならL1番地にジャンプする命令を実行すればよい。

【4bit 符号なしの場合のフラグの変化】

	A = Bの場合	A > Bの場合1	A > Bの場合2	A < Bの場合
A	9 1001	9 1001	9 1001	7 0111
- B	- 9 + 0111	- 7 + 1001	- 1 + 1111	- 9 + 0111
	0 10000	2 10010	8 11000	-2 1110
	CY=0	CY=0	CY=0	CY=1
	S=0	S=0	S=1	S=1
	Z=1	Z=0	Z=0	Z=0

【4bit 符号あり (2の補数表現) の場合のフラグの変化】

	A = Bの場合	A > Bの場合1	A > Bの場合2	A < Bの場合
A	6 0110	6 0110	6 0110	2 0010
- B	- 6 + 1010	- 2 + 1110	- (-1) + 0001	- 6 + 1010
	0 10000	4 10100	7 0111	-4 1100
	CY=0	CY=0	CY=1	CY=1
	S=0	S=0	S=0	S=1
	Z=1	Z=0	Z=0	Z=0

【4bit コード表 (参考)】

2進数	10進数 (符号なし)	10進数(符号あり)
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

4.5. 制御入出力信号

【制御入力信号】

$\overline{\text{RESET IN}}$ (36 番ピン) ... 0 を入力すると、CPU をリセットして、0 番地から命令を実行
(上にバーが付いているときは、負論理なので 0 を入力すると RESET される。)

【制御出力信号】

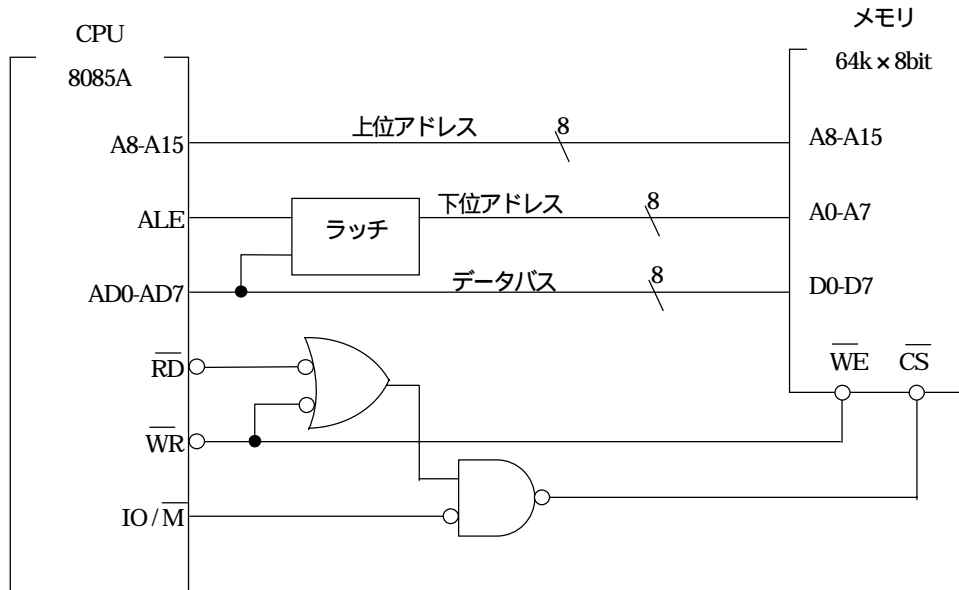
$\text{IO} / \overline{\text{M}}$ (34 番ピン) ... 入出力装置(I/O)とメモリの区別をする
1 が出力されたときは、IO に対する読み書き
0 が出力されたときは、メモリーに対する読み書き

$\overline{\text{RD}}$ (32 番ピン) ... 0 が出力されたとき、CPU が IO あるいはメモリーからデータを読み込みたいので、データを出力してほしいことを意味する

$\overline{\text{WR}}$ (31 番ピン) ... 0 が出力されたとき、CPU が IO あるいはメモリーにデータを出力したので、データを読み込んでほしいことを意味する

4.6. CPU とメモリの接続

(1) CPU8085A と 64k 番地 × 8bit メモリとの接続



メモリ \overline{WE} ... 0:書き込み、1:読み込み

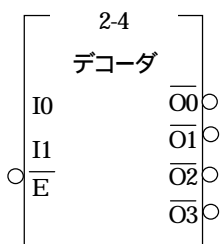
\overline{CS} ... 0を入力したとき、指定された番地のメモリとデータバスを電氣的に接続

メモリ \overline{WE} CPU \overline{WR}

メモリ \overline{CS} CPU $\overline{M \cdot (RD + WR)}$

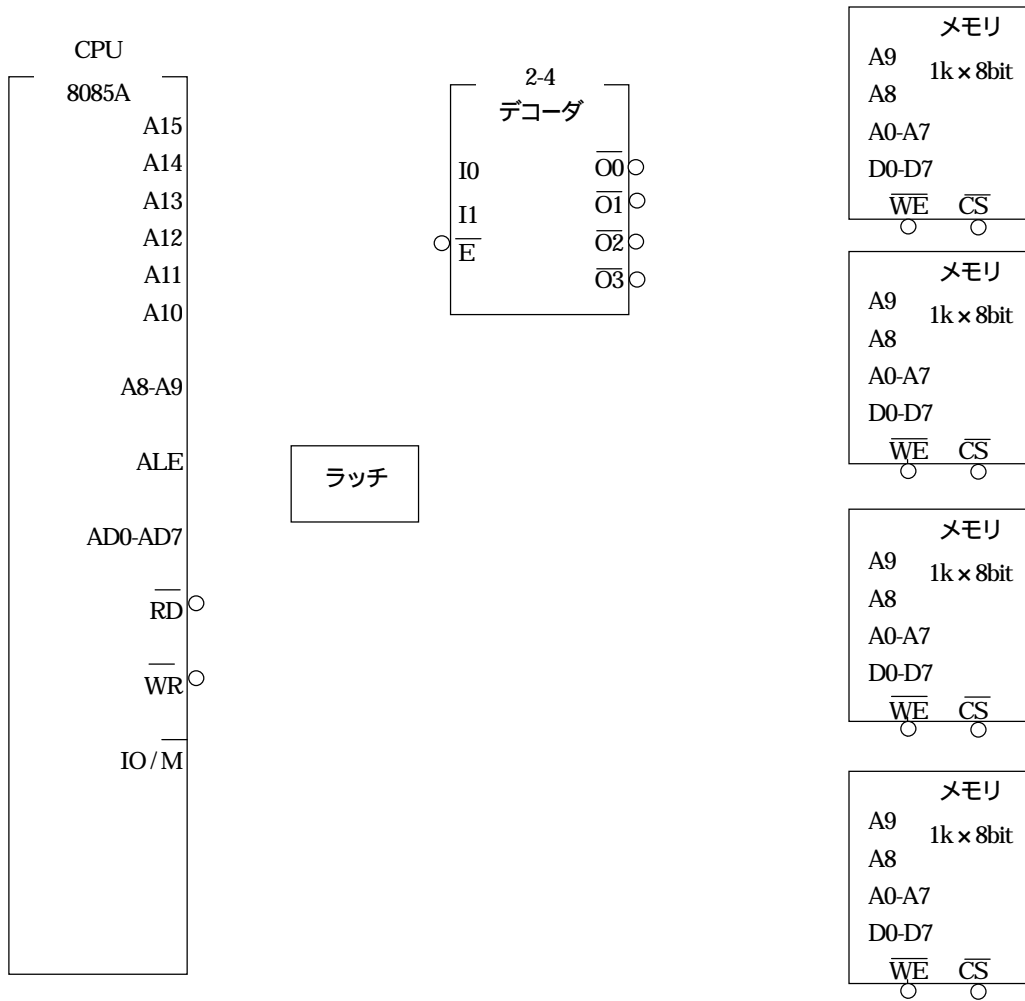
メモリに対する操作でかつ、read または write のとき、Chip Select (CS) すればよい。

(2) 2-4 デコーダ (複数のメモリを使用する際に使用するアドレスデコーダなどに使用) 以下の真理値表のように動作するデコーダ回路を示せ。

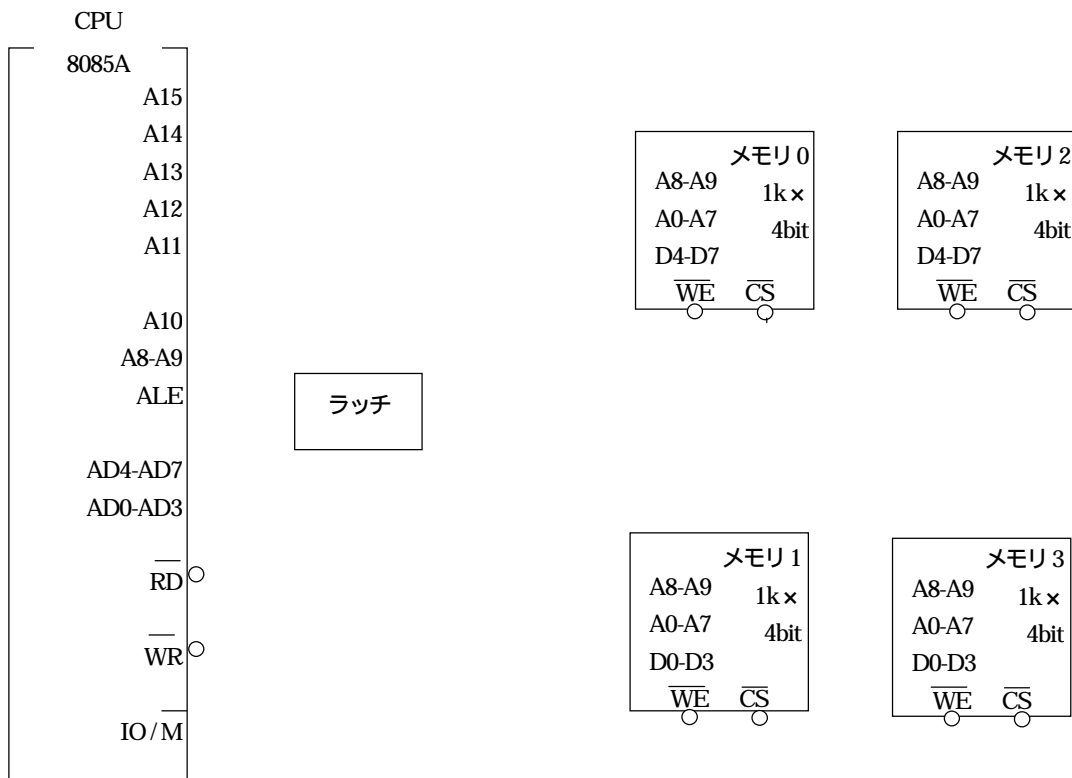


入力信号			出力信号			
\overline{E}	I1	I2	$\overline{O3}$	$\overline{O2}$	$\overline{O1}$	$\overline{O0}$
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1

(3) CPU8085A と 1k 番地 × 8bit × 4 個のメモリとの接続



(4) 同様に CPU8085A と 1k 番地 × 4bit × 4 個のメモリとの結線を行え。



4.7. マシンサイクル ([2] p.39~p.40)

1つの命令 ... 数個の**マシンサイクル**

マシンサイクル ... 数個の**ステート** (クロックパルス1周期分)

(1) マシンサイクルの種類 (一部)

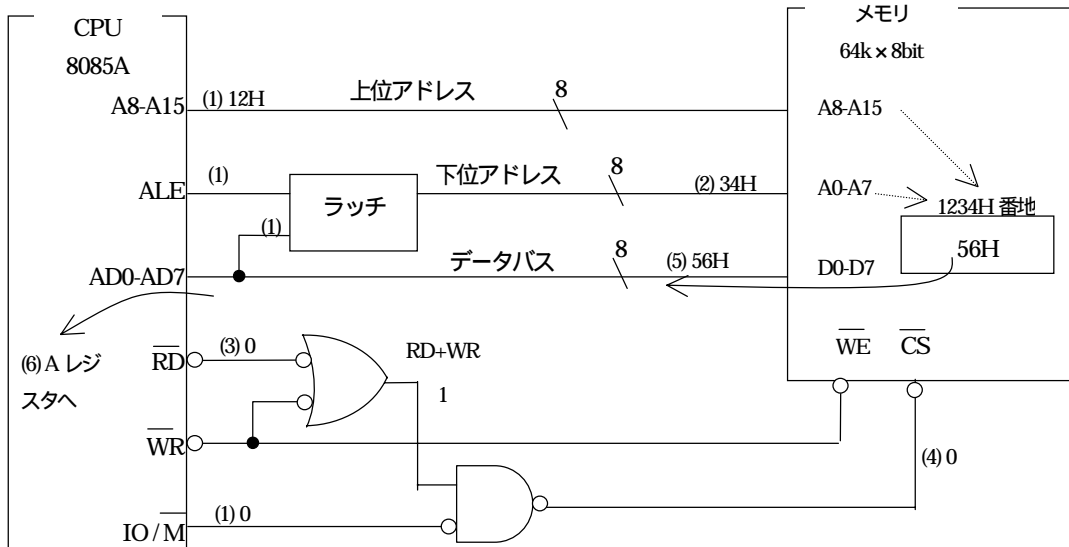
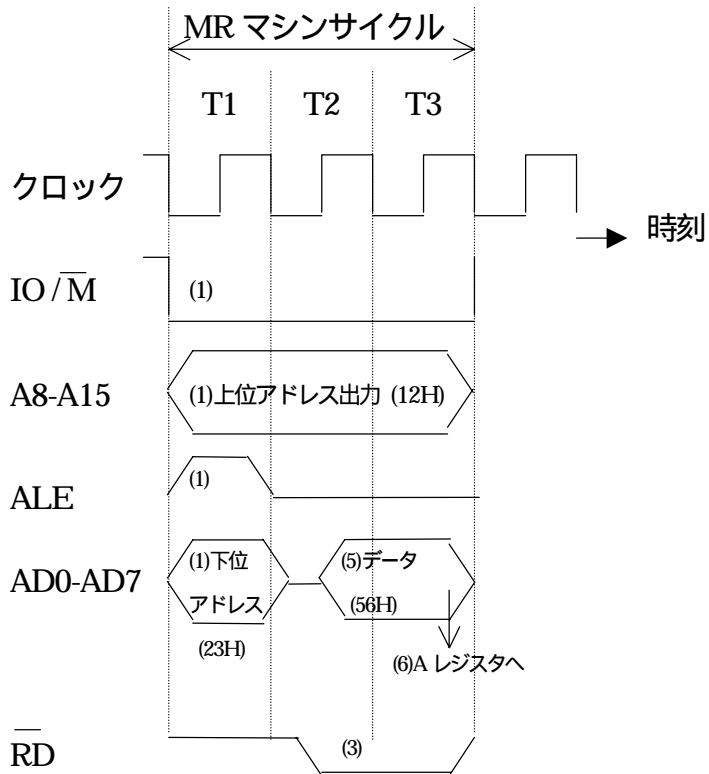
OPECODE FETCH (OF)	命令データの読み込み
MEMORY READ (MR)	(メモリ) (Aレジスタ)
MEMORY WRITE (MW)	(Aレジスタ) (メモリ)
I/O READ (IOR)	(入出力装置) (Aレジスタ)
I/O WRITE (IOW)	(Aレジスタ) (入出力装置)

(2) 1234H番地のメモリの内容(56H)をCPU内のAレジスタに転送する場合

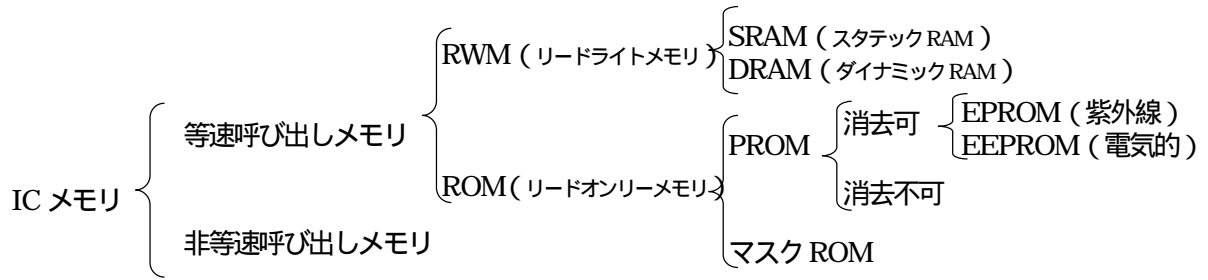
この命令は、OFマシンサイクルとMRマシンサイクルからなる。また、MRマシンサイクルは以下の3つのステート(クロック3周期)からなる。以下の図はMRマシンサイクルを表している。

【MRマシンサイクル】

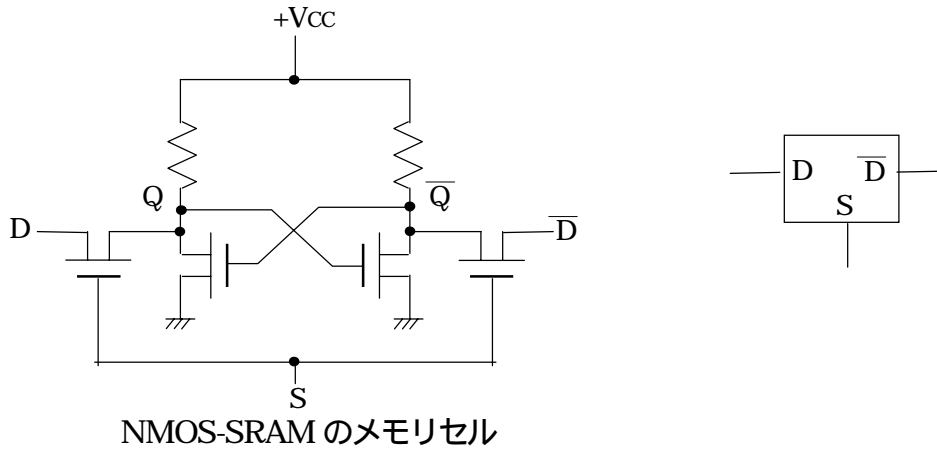
- 最初のステート(T1) (1) $\overline{IO/\overline{M}}$ が0になり、メモリに対する操作であることが指定される。A8-A15には上位アドレス(12H)が出力される。またALEが1になり、AD0-AD7には下位アドレス(34H)が出力され、ラッチに記憶される。
- (2) 上位アドレスとラッチに記憶された下位アドレスがメモリに与えられる。
- 2番目のステート(T2) (3) \overline{RD} が0になる。
- (4) \overline{CS} に0が与えられる。
- (5) 1234H番地のメモリとデータバスが電氣的に接続され、データバスにメモリの内容(56H)が出力される。
- 3番目のステート(T3) (6) データバスの値(56H)がCPU内のAレジスタに取り込まれる。



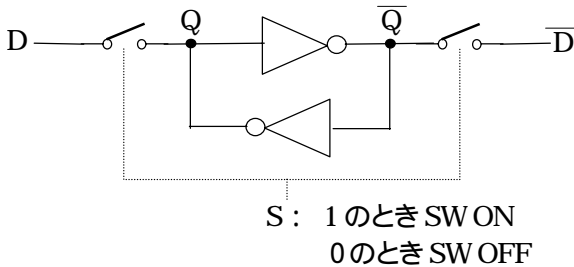
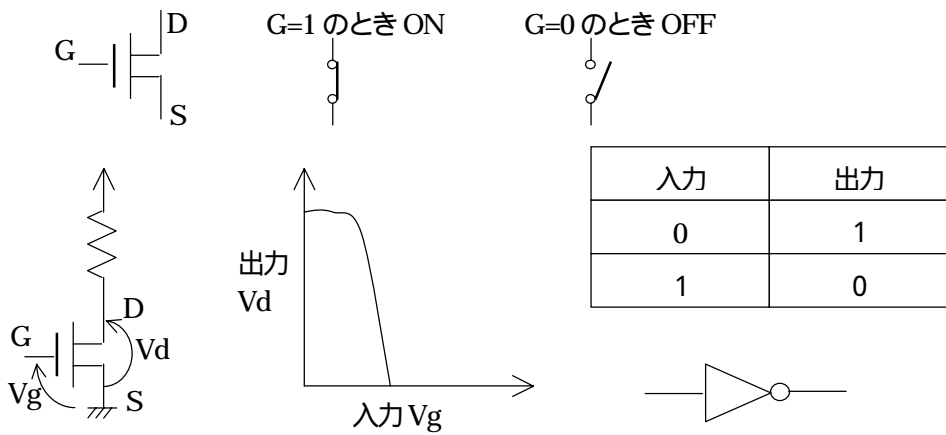
4.8. メモリ ([2] p.60~p.73)



(1) SRAM



エンハンスメント型 NMOS FET

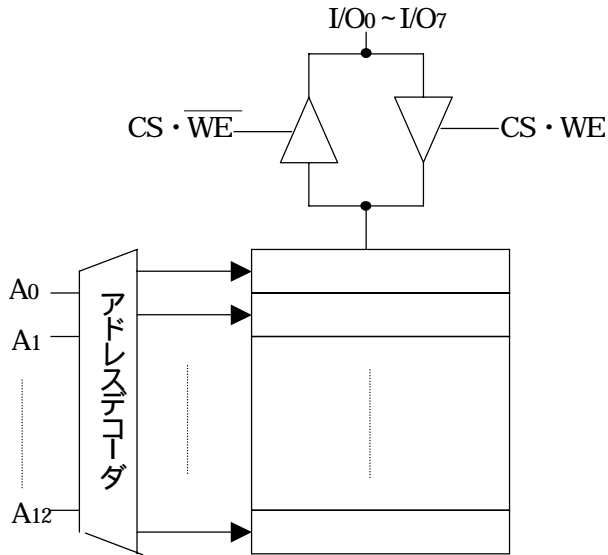


S=0 のとき
2 状態 (Q=1, Q-bar=0 または Q=0, Q-bar=1) の
いずれかを保持

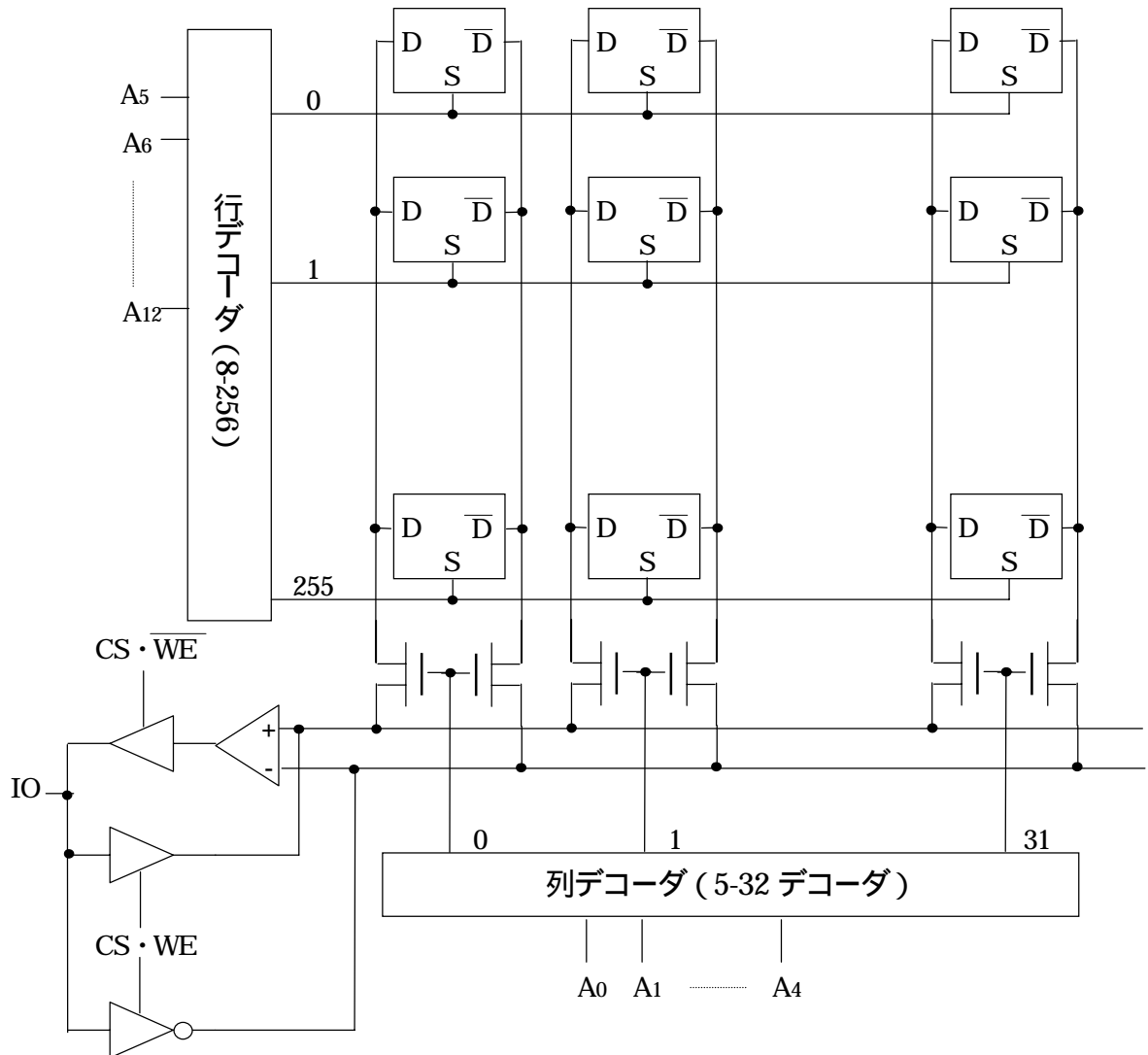
Write 時
S=1 にして D=1, D-bar=0 Q=1
 D=0, D-bar=1 Q=0

Read 時
S=1 にして D > D-bar 1 と判定
 D < D-bar 0 と判定

(2) SRAMの構成例 (8192ワード×8bit構成)



13-8192 アドレスデコーダ



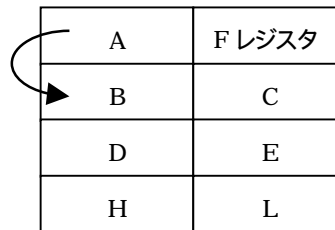
5. アセンブリ言語 (Z 8 0)

5.1. 命令形式

(1) 1バイト命令

例： A レジスタの内容を B レジスタにコピー

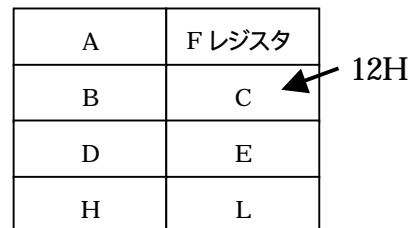
機械語コード ニーモニック メモ
47H LD B,A B A



(2) 2バイト命令

例： C レジスタに 12H を代入

機械語コード ニーモニック メモ
0EH 12H LD C,12H C 12H



(3) 3バイト命令

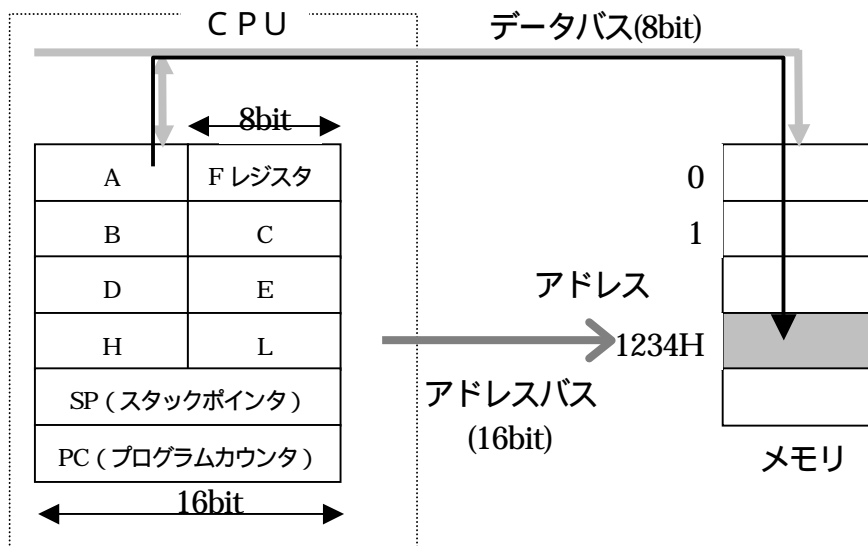
例： A レジスタの内容を 1234H 番地にコピー

機械語コード ニーモニック メモ
32H 34H 12H LD (1234H),A (1234H) A
B1 B2 B3

32H オペレーションコード (オペコード) B1

34H 12H オペランド B2 B3

(8085A や Z80 の場合、アドレスは下位アドレス上位アドレスの順に指定する)



5.2. アセンブリ言語

アセンブリ言語 ... 主として機械語と 1対1 に対応するニーモニックの集まり

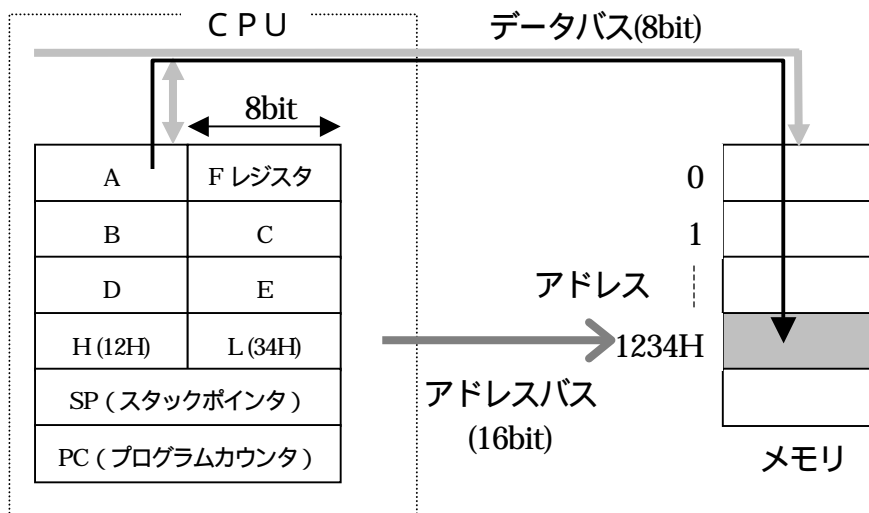
アセンブラ ... アセンブリ言語を機械語に変換するプログラム (コンパイラ)

5.3. 命令の種類 (CPU Z80 の場合)

(1) データ転送命令 1

r \	A	B	C	D	E	H	L	(HL)	メモ
LDA, r	7F	78	79	7A	7B	7C	7D	7E	A ← r
LDB, r	47	40	41	42	43	44	45	46	B ← r
LDC, r	4F	48	49	4A	4B	4C	4D	4E	C ← r
LDD, r	57	50	51	52	53	54	55	56	D ← r
LDE, r	5F	58	59	5A	5B	5C	5D	5E	E ← r
LDH, r	67	60	61	62	63	64	65	66	H ← r
LDL, r	6F	68	69	6A	6B	6C	6D	6E	L ← r
LD(HL), r	77	70	71	72	73	74	75		HL ← r
LD r, B ₂	3E	06	0E	16	1E	26	2E	36	r ← 1バイトの数値 B ₂

	ニーモニック	コメント	マシンコード
例 1	LDB, C	; B C	41H
例 2		; C D	
例 3	LDH, A	;	
例 4		;	68H
例 5	LDB, 12	; B 12	06H 0CH
例 6	LDB, 0CH	; B 0CH	06H 0CH
例 7	LDD, 32H	;	
例 8	LDH, 12H	; H 12H	26H 12H
	LDL, 34H	; L 34H	2EH 34H
	LD(HL), A	; (HL) A	77H
		; 結局, (1234H)	A と同等



	ニーモニック	コメント	マシンコード
例 9	LD H,12H	; H 12H	26H 12H
	LD L,34H	; L 34H	2EH 34H
	LD C,(HL)	; C (HL)	4EH
		; 即ち、C (1234H)、1234H 番地の内容を C レジスタにコピー	

例 10 8100H 番地の内容を C レジスタにコピー

;
;
;

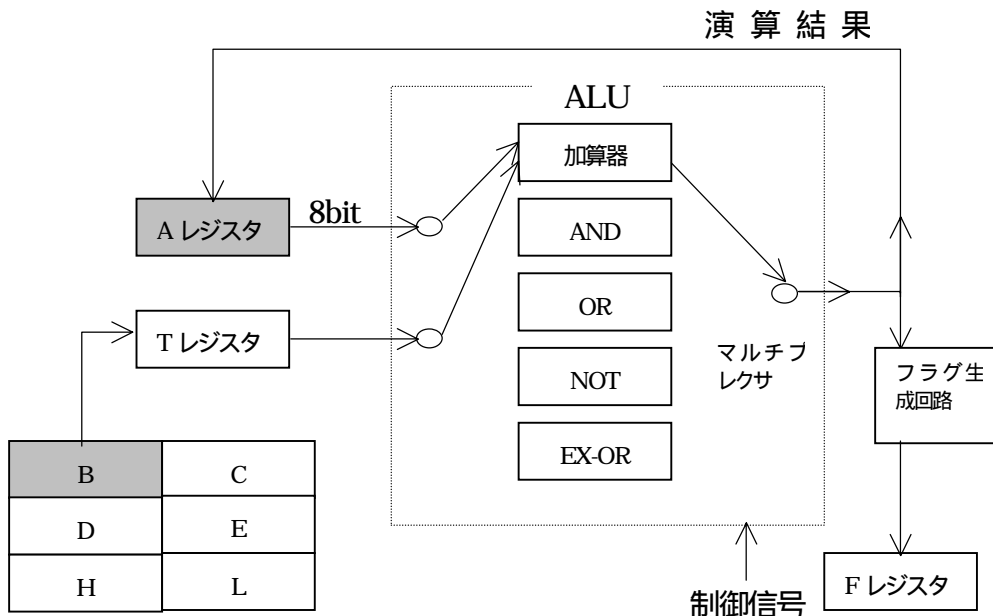
例 11 B レジスタの内容を 8200H 番地にコピー

;
;
;

(2) 演算命令

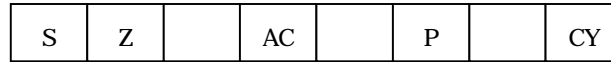
r \	A	B	C	D	E	H	L	(HL)	メモ
ADDA, r	87	80	81	82	83	84	85	86	$A \leftarrow A+r$
ADCA, r	8F	88	89	8A	8B	8C	8D	8E	$A \leftarrow A+r+CY$
SUB r	97	90	91	92	93	94	95	96	$A \leftarrow A-r$
SBCA, r	9F	98	99	9A	9B	9C	9D	9E	$A \leftarrow A-r-CY$
AND r	A7	A0	A1	A2	A3	A4	A5	A6	$A \leftarrow A \text{ AND } r$
XOR r	AF	A8	A9	AA	AB	AC	AD	AE	$A \leftarrow A \text{ Ex-OR } r$
OR r	B7	B0	B1	B2	B3	B4	B5	B6	$A \leftarrow A \text{ OR } r$
CP r	BF	B8	B9	BA	BB	BC	BD	BE	A-r フラグのみ変化
INC r	3C	04	0C	14	1C	24	2C	34	$r \leftarrow r+1$
DEC r	3D	05	0D	15	1D	25	2D	35	$r \leftarrow r-1$

	ニーモニック	コメント	マシンコード
例 1	ADDA,B	; A A+B	80H



例2 ニーモニック コメント マシンコード
 ADC A,C ; A A + C + CY(キャリーフラグ) 89H

Fレジスタ



MSB

LSB

```

    3 5
  + 1 7
  ---
    5 2
  3+1+1(CY)=5
  
```

```

    3 F F
  + 2 0 2
  ---
    6 0 1
  3+2+1(CY)=6
  
```

例3 SUB D ; A A - D H

例4 SBC A,E ; A A - E - CY H

```

    3 5
  - 1 7
  ---
    1 8
  3 - 1 - 1(CY)=1
  
```

```

    3 0 1
  - 2 0 2
  ---
    0 F F
  3 - 2 - 1(CY)=0
  
```

例5 AND H ; A A AND H H

例6 OR L ; A A OR L H

例7 XOR B ; A A ⊕ B H
 排他的論理和(EX-OR)

各ビットごとの論理演算

<pre> 1 1 0 0 AND 1 0 1 0 --- 1 0 0 0 </pre>	<pre> 1 1 0 0 OR 1 0 1 0 --- 1 1 1 0 </pre>	<pre> 1 1 0 0 ⊕ 1 0 1 0 --- 0 1 1 0 </pre>
--	--	---

例8 比較命令 (結果を保存しない. フラグのみ変化)
 CP D ; A - D H
 if(_____){ }
 ここに使う

例9 INC E ; E E + 1 H

例10 DEC H ; H H - 1 H

例11 ADD A, 23H
 ; A A + 23H
 C6 23

例12 SBC A, 0F0H
 ; A A - 0FH - CY
 — —

例13 CP 12
 ; A - 12 (10進)
 — —

Z80	コード	メモ
ADD A, B ₂ ADC A, B ₂	C6 CE	A ← A + B ₂ A ← A + B ₂ + CY
SUB B ₂ SBC A, B ₂	D6 DE	A ← A - B ₂ A ← A - B ₂ - CY
AND B ₂ XOR B ₂ OR B ₂	E6 EE F6	A ← A AND B ₂ A ← A Ex-OR B ₂ A ← A OR B ₂
CP B ₂	FE	A - B ₂ フラグのみ変化

(3) ペアレジスタを用いた命令

例1 LD HL, 1234
 ; HL 1234 (10進)

例2 LD BC, 8000H
 ; BC 8000H
 — — —

	r ₁ r ₂	BC	DE	HL	SP	メモ
LD r ₁ r ₂ , B ₃ B ₂		01	11	21	31	r ₁ r ₂ ← B ₃ B ₂
ADD HL, r ₁ r ₂		09	19	29	39	HL ← HL + r ₁ r ₂
INC r ₁ r ₂		03	13	23	33	r ₁ r ₂ ← r ₁ r ₂ + 1
DEC r ₁ r ₂		0B	1B	2B	3B	r ₁ r ₂ ← r ₁ r ₂ - 1

例3 ADD HL, BC ; HL HL + BC —

例4 ; HL HL + DE —

例5 INC BC ; BC BC + 1 —

例6 ; HL HL - 1 —

例7 (1234H 番地) A

(方法1) LD BC, 1234H ; BC 1234H
 LD (BC), A ; (BC) A

(方法2) LD DE, 1234H ; DE 1234H
 LD (BC), A ; (DE) A

(方法3) LD HL, 1234H ; HL 1234H
 LD (HL), A ; (HL) A

(方法4) LD (1234H), A ; (1234H) A

Z80	コード	メモ
LD (BC), A	02	(BC) ← A
LD A, (BC)	0A	A ← (BC)
LD (DE), A	12	(DE) ← A
LD A, (DE)	1A	A ← (DE)
LD (B ₃ B ₂), A	32	(B ₃ B ₂) ← A
LD A, (B ₃ B ₂)	3A	A ← (B ₃ B ₂)

例8 A (1234H 番地)
(方法1)

(方法2)

(方法3)

(方法4)

例9 LD (1234H), HL
; (1234H) L
; (1235H) H

例10 LD HL, (1234H)
; L (1234H)
; H (1235H)

例11 EX (SP), HL
; (SP) L 値の交換 (入れ替え)
; (SP+1) H

Z80	コード	メモ
LD (B ₃ B ₂), HL	22	(B ₃ B ₂) ← L (B ₃ B ₂ +1) ← H
LD HL, (B ₃ B ₂)	2A	L ← (B ₃ B ₂) H ← (B ₃ B ₂ +1)
EX (SP), HL	E3	(SP) ↔ L (SP+1) ↔ H
EX DE, HL	EB	DE ↔ HL
JP (HL)	E9	PC ← HL
LD SP, HL	F9	SP ← HL

例12 JP (HL) ; PC HL HL 番地にジャンプ

(4) ジャンプ命令

例1 JP 1234H ; 1234H 番地にジャンプ
C3 34 12

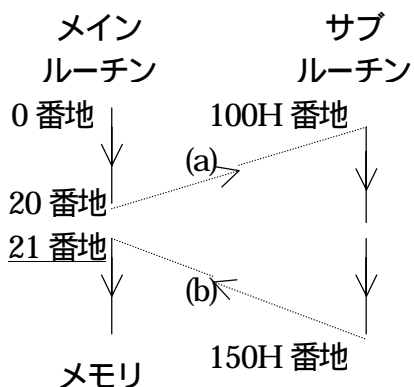
例2 条件付きジャンプ (分岐命令)
JP 条件, 1234H ; 条件を満たせば 1234H 番地に
ジャンプ

条件: Z Z フラグ=1 直前の演算結果が 0 なら
NZ Z フラグ=0 直前の演算結果が 0 でない
C CY フラグ=1 例えば直前の演算結果が負
(符号なし引き算の場合)
NC CY フラグ=0 例えば直前の演算結果が正または 0 (符号なし引き算の場合)
M S フラグ=1 例えば直前の演算結果が負 (2 の補数引き算の場合)
P S フラグ=0 例えば直前の演算結果が正または 0 (2 の補数引き算の場合)
PE パリティフラグ=1 直前の演算結果の 1 の数が偶数個
PO パリティフラグ=0 直前の演算結果の 1 の数が奇数個

Z80	コード
JP B ₃ B ₂	C3
JP Z, B ₃ B ₂	CA
JP NZ, B ₃ B ₂	C2
JP C, B ₃ B ₂	DA
JP NC, B ₃ B ₂	D2
JP PE, B ₃ B ₂	EA
JP PO, B ₃ B ₂	E2
JP M, B ₃ B ₂	FA
JP P, B ₃ B ₂	F2

(5) コール命令、リターン命令

例1



(a) サブルーチンコール時

CALL 100H
CD 00 01

(b) リターン時

RET
C9

例2 条件付きコール命令

CALL 条件, 1234H
; 条件を満たせば 1234H 番地に
サブルーチンコール

条件 : Z	Z フラグ=1	直前の演算結果が 0 なら
NZ	Z フラグ=0	直前の演算結果が 0 でない
C	CY フラグ=1	例えば直前の演算結果が負 (符号なし引き算の場合)
NC	CY フラグ=0	例えば直前の演算結果が正または 0 (符号なし引き算の場合)
M	S フラグ=1	例えば直前の演算結果が負 (2 の補数引き算の場合)
P	S フラグ=0	例えば直前の演算結果が正または 0 (2 の補数引き算の場合)
PE	パリティフラグ=1	直前の演算結果の 1 の数が偶数個
PO	パリティフラグ=0	直前の演算結果の 1 の数が奇数個

例2 条件付きリターン命令

RET 条件 ; 条件を満たせばリターン

コール命令

Z80	コード
CALL B ₃ B ₂	CD
CALL Z, B ₃ B ₂	CC
CALL NZ, B ₃ B ₂	C4
CALL C, B ₃ B ₂	DC
CALL NC, B ₃ B ₂	D4
CALL PE, B ₃ B ₂	EC
CALL PO, B ₃ B ₂	E4
CALL M, B ₃ B ₂	FC
CALL P, B ₃ B ₂	F4

リターン命令

Z80	コード
RET	C9
RET Z	C8
RET NZ	C0
RET C	D8
RET NC	D0
RET PE	E8
RET PO	E0
RET M	F8
RET P	F0

フラグレジスタ

S	Z	0	AC	0	P	1	CY
MSB				LSB			

5.4. プログラム作成例

1 + 2 + ... + 20 を求めて結果を SUM 番地 (9000H 番地) に格納する。

(1) 高級言語 (C 言語) の場合

```
#include <stdio.h>
void main(void)
{
    int A, B;
    A = 0;
    for(B = 20; B > 0; B - -){
        A = A + B;
    }
}
```

(2) まず、全体の処理概要を段階的に箇条書きにしてみる

初期化

1. A を 0 に初期化
2. B に 20 を代入

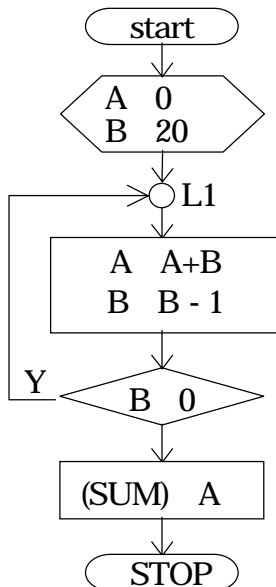
1 + 2 + ... + 20 を求める

3. A に A+B を代入
4. B を 1 つ減らす
5. B が 0 になるまで 3, 4 を繰り返す

結果を格納

6. SUM 番地に A をコピー

(3) フローチャートを作成



(4) コメントを作成

アドレス	マシン語	ニーモニック ; コメント
		<pre> ;A 0 ;B 20 L1: ;A A + B ;B B - 1 ;if(B 0) goto L1 ;if(Z フラグ=0) goto L1 ;(SUM) A ;stop SUM: ;SUM 番地 (9000H 番地) に0を入 ; しておく </pre>

(5) コメントに従ってニーモニックを書く

アドレス	マシン語	ニーモニック ; コメント
		<pre> ORG 8000H ;8000H 番地からプログラムを記述 LD A,0 ;A 0 LD B,20 ;B 20 L1: ADD A,B ;A A + B DEC B ;B B - 1 JP NZ, L1 ;if(B 0) goto L1 ;if(Z フラグ=0) goto L1 LD (SUM), A ;(SUM) A HALT ;stop ORG 9000H SUM: DB 0 ;SUM 番地 (9000H 番地) に0を入 ; しておく END </pre>

(6) ニーモニックをマシンコードに変換 (不明なアドレスは空白にしておく)

アドレス	マシン語	ニーモニック ; コメント
		<pre> ORG 8000H ;8000H 番地からプログラムを記述 LD A,0 ;A 0 LD B,20 ;B 20 L1: ADD A,B ;A A + B DEC B ;B B - 1 JP NZ, L1 ;if(B 0) goto L1 ;if(Z フラグ=0) goto L1 LD (SUM), A ;(SUM) A HALT ;stop ORG 9000H SUM: DB 0 ;SUM 番地 (9000H 番地) に0を入 ; しておく END </pre>
	3E 00	LD A,0
	06 14	LD B,20
	80	L1: ADD A,B
	05	DEC B
	C2 _ _	JP NZ, L1
	32 00 90	LD (SUM), A
	76	HALT
	00	SUM: DB 0
		END

(7) アドレスを記入する

アドレス	マシン語	ニーモニック ; コメント
8000H	3E 00	ORG 8000H ; 8000H 番地からプログラムを記述
8002H	06 14	LD A, 0 ; A 0
8004H	80	L1: ADD A, B ; A A + B
8005H	05	DEC B ; B B - 1
8006H	C2 _ _	JP NZ, L1 ; if(B 0) goto L1 ; if(Z フラグ=0) goto L1
8009H	32 00 90	LD (SUM), A ; (SUM) A
800CH	76	HALT ; stop
9000H	00	ORG 9000H SUM: DB 0 ; SUM 番地 (9000H 番地) に 0 を入 ; れておく
		END

(8) マシン語の空白のアドレスを埋めて、完成

アドレス	マシン語	ニーモニック ; コメント
8000H	3E 00	ORG 8000H ; 8000H 番地からプログラムを記述
8002H	06 14	LD A, 0 ; A 0
8004H	80	L1: ADD A, B ; A A + B
8005H	05	DEC B ; B B - 1
8006H	C2 <u>04 80</u>	JP NZ, L1 ; if(B 0) goto L1 ; if(Z フラグ=0) goto L1
8009H	32 00 90	LD (SUM), A ; (SUM) A
800CH	76	HALT ; stop
9000H	00	ORG 9000H SUM: DB 0 ; SUM 番地 (9000H 番地) に 0 を入 ; れておく
		END

(9) 説明書に従って、マシンコードをメモリに格納

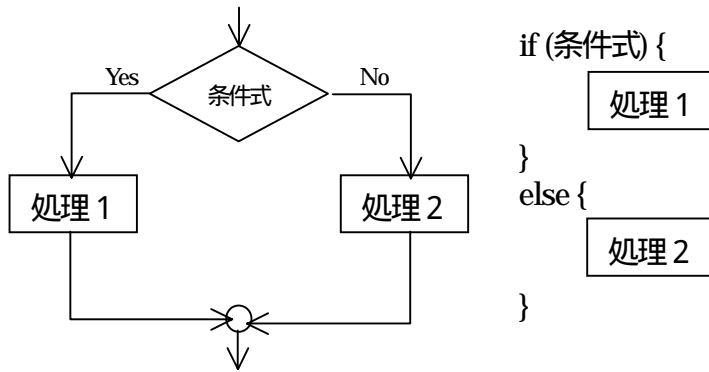
(10) 実行して、9000H の内容が 1 から 20 までの和になっているか確認

(11) 1から3までの和を求めるプログラムに変更してトレース実行してみる
 プログラムで使用しているレジスタ、フラグ、メモリ内容を順次記録する。

アドレス	マシンコード	ニーモニック	A	B	Z	9000H 番地	PC
初期状態			00	00	0	00	8000
8000	3E 00	LD A, 0 ; A 0	00	00	0	00	8002
8002	06 03	LD B, 3 ; B 3	00	03	0	00	8004
8004	80	L1: ADD A, B ; A A + B	03	03	0	00	8005
		DEC B ; B B - 1					
		JP NZ, L1 ; if(B 0) goto L1					
		L1: ADD A, B ; A A + B					
		DEC B ; B B - 1					
		JP NZ, L1 ; if(B 0) goto L1					
		L1: ADD A, B ; A A + B					
		DEC B ; B B - 1					
		JP NZ, L1 ; if(B 0) goto L1					
		LD (SUM), A ; (SUM) A					
		HALT ; stop					

5.5. 条件分岐 (if 文)

(1) C 言語の場合



(2) 条件とフラグ

A レジスタの内容と r レジスタの内容を比較する場合

A - r を計算 CP r フラグが以下のように変化

	条件	Z	CY	S	ニーモニック
	A = B	1			JP Z, PROC1
	A != B	0			JP NZ, PROC1
符号なし (0 ~ 255)	A > B		0		JP NC, PROC1
	A < B		1		JP C, PROC1
符号あり (-128 ~ 127)	A > B			0	JP P, PROC1
	A < B			1	JP M, PROC1

(3) アセンブラによる if 文の実現 (方法 1)

<pre> if (A == B) { 処理 1 } else { 処理 2 } </pre>	<pre> CP B ; A - B JP Z, PROC1 ; if(A != B) { (Zフラグが1なら PROC1 にジャンプ) 処理 2 } JP ENDIF ; } PROC1: ; else { 処理 1 } ENDIF: ; } </pre>
---	---

(4) アセンブラによる if 文の実現 (方法 2)

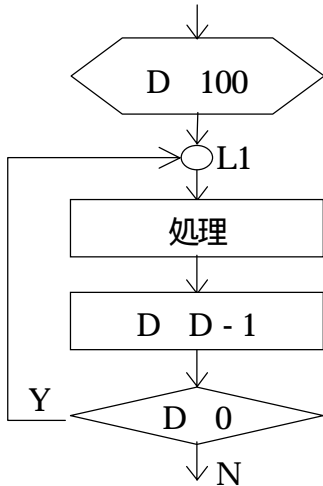
<pre> if (A == B) { 処理 1 } else { 処理 2 } </pre>	<pre> CP B ; A - B JP NZ, PROC2 ; if(A == B) { (Zフラグが0なら PROC2 にジャンプ) 処理 1 } JP ENDIF ; } PROC2: ; else { 処理 2 } ENDIF: ; } </pre>
---	--

(5) 練習問題

D1番地(8100H)の内容がD2番地(8101H)の内容より小さければD3番地(8102H)番地に1を、小さくなければ0を代入するプログラムを作成し、トレース実行によって各レジスタ等の値を調べ、動作を確認せよ。なお、D1,D2番地には符号ありの整数が入っているものとする。

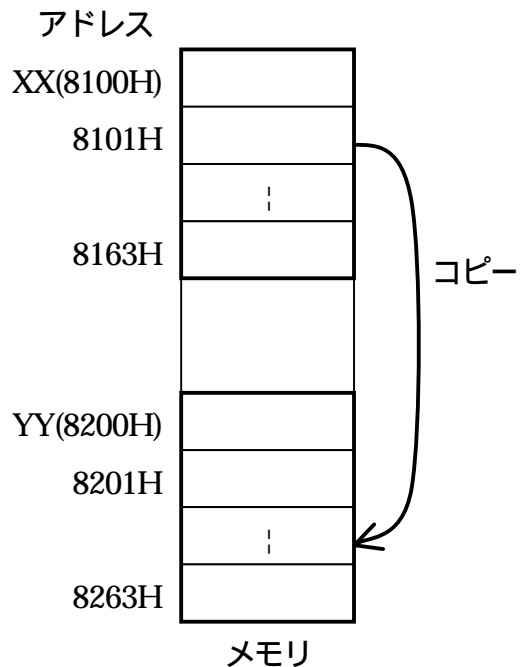
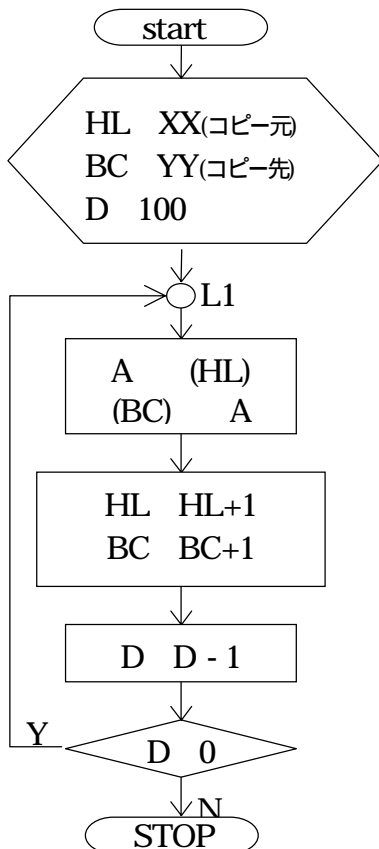
5.6. 指定回数ループ

(1) 100回ループ



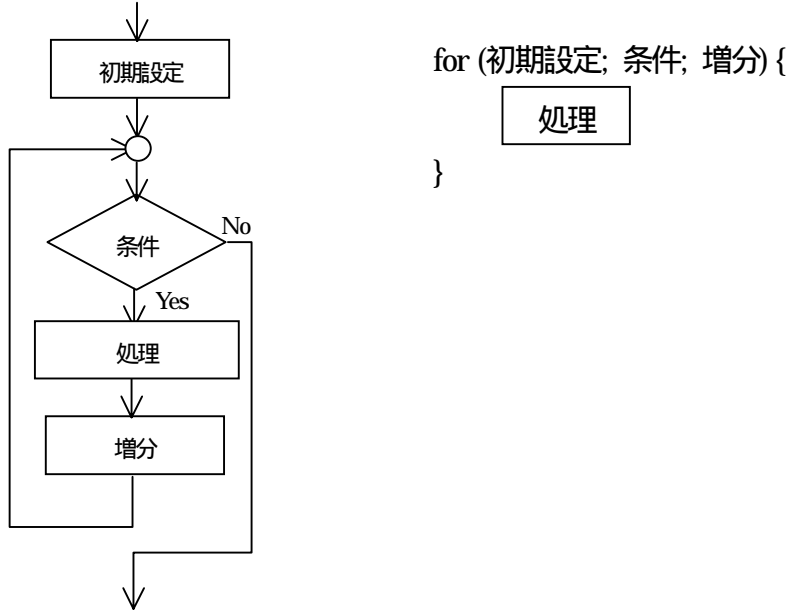
(2) 練習問題

XX(8100H番地)から始まる100個のデータをYY(8200H番地)以降にコピーするプログラムを作成せよ。

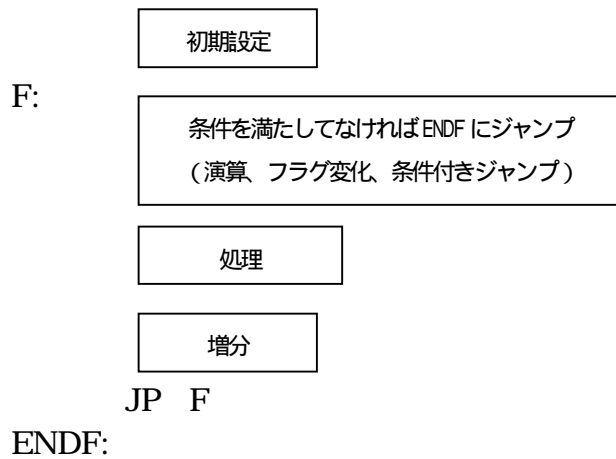


5.7. for文(繰り返し処理2) P.53

(1) C言語の場合



(2) アセンブラによる for 文の実現

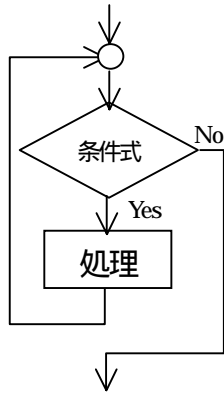


(3) 練習問題

1+3+...+19 を計算し、結果を 8100H に格納するプログラムを作成せよ。

5.8. while 文

(1) 書式



```
while (条件式) {  
    処理  
}
```

(2) アセンブラによる while 文の実現

W:

条件を満たしてなければENDWにジャンプ
(演算、フラグ変化、条件付きジャンプ)

処理

JP W

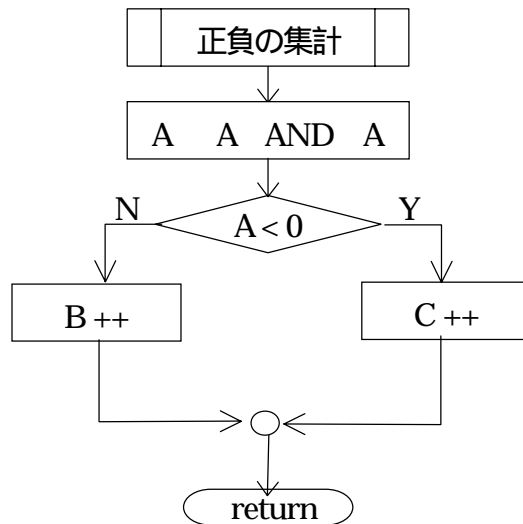
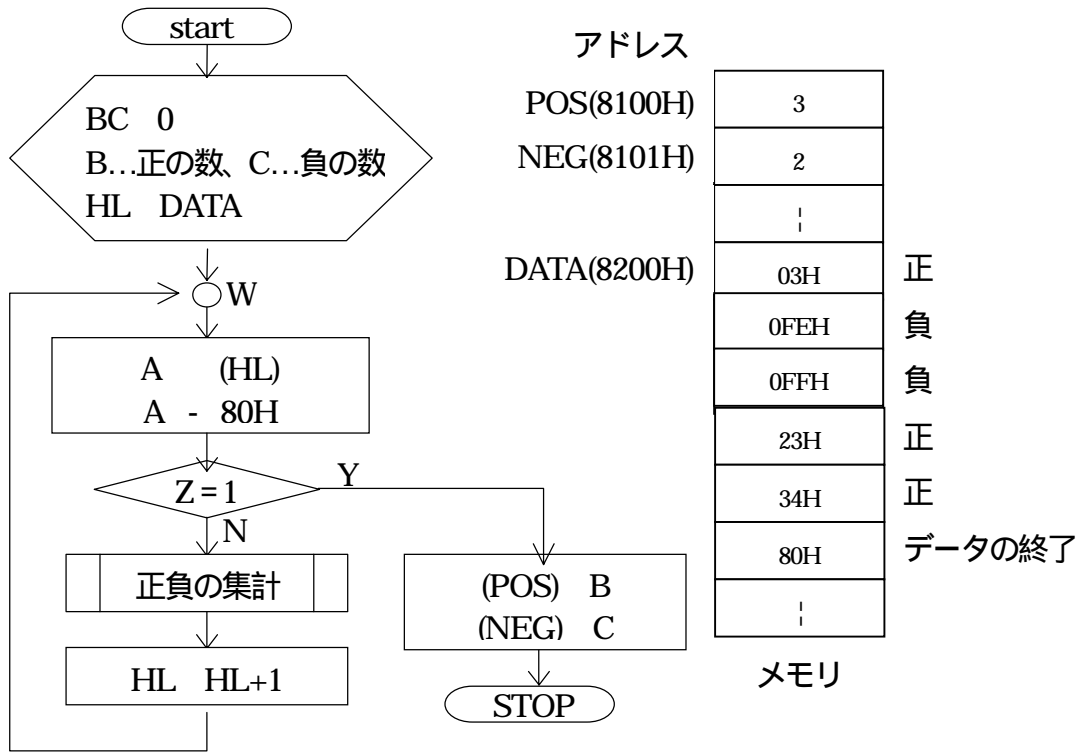
ENDW

5.9. 練習問題

DATA(8200H)番地以降のデータの正または0の数を POS(8100H)番地に、負の数を NEG(8101H)番地に格納する。ただし、80Hはデータの終了を意味するものとする。

(ヒント)

DATA(8200H)番地以降のデータの正または0の数を POS(8100H)番地に、負の数を NEG(8101H)番地に格納する。ただし、80Hはデータの終了を意味するものとする。



5.10. bit 演算

(1) bit の取り出し

下から 3bit 目と 4bit 目を取り出す。

$$\begin{array}{r}
 1100 \boxed{1010} \text{ (元データ)} \\
 \text{AND } 0000 \boxed{1100} \\
 \hline
 0000 \boxed{1000}
 \end{array}
 \quad
 \begin{array}{l}
 x \cdot 1 = x \\
 x \cdot 0 = 0
 \end{array}$$

(2) bit のセット

下から 2bit 目、3bit 目、4bit 目を 1 にセットする。

$$\begin{array}{r}
 1100 \boxed{1010} \text{ (元データ)} \\
 \text{OR } 0000 \boxed{1110} \\
 \hline
 1100 \boxed{1110}
 \end{array}
 \quad
 \begin{array}{l}
 x + 1 = 1 \\
 x + 0 = x
 \end{array}$$

(3) bit のリセット

上位 4bit を 0 にリセットする。

$$\begin{array}{r}
 \boxed{1100} 1010 \text{ (元データ)} \\
 \text{AND } 0000 \boxed{1111} \\
 \hline
 \boxed{0000} 1010
 \end{array}
 \quad
 \begin{array}{l}
 x \cdot 0 = 0 \\
 x \cdot 1 = x
 \end{array}$$

(4) bit の反転

下位 4bit を反転する。

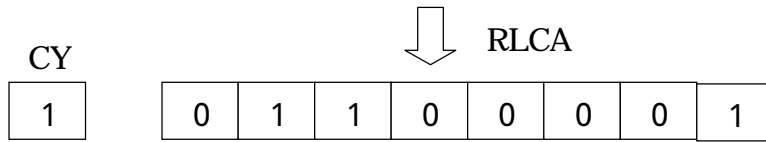
$$\begin{array}{r}
 1100 \boxed{1010} \text{ (元データ)} \\
 \text{EX-OR } 0000 \boxed{1111} \\
 \hline
 1100 \boxed{0101}
 \end{array}
 \quad
 \begin{array}{l}
 x \oplus 0 = x \\
 x \oplus 1 = \bar{x}
 \end{array}$$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

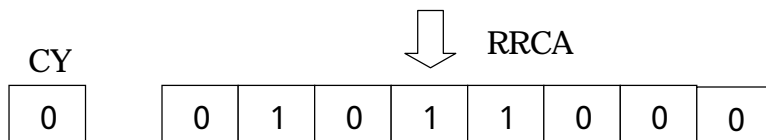
(例題) 8100H 番地の下位 4bit と 8101H 番地の下位 4bit が一致すれば 8102 番地に 1 を、一致しなければ 0 を格納するプログラムを作成せよ。

5.11. 回転シフト命令

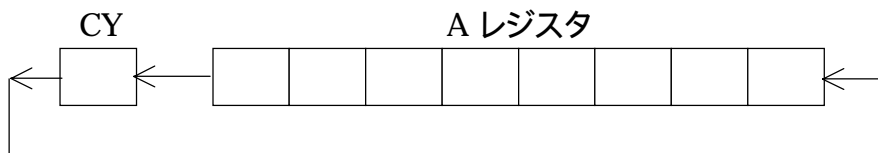
(1) RLCA



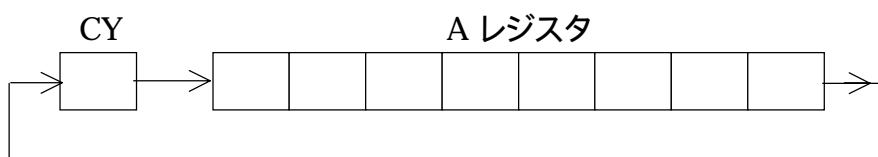
(2) RRCA



(3) RLA



(4) RRA



(5) シフト演算を用いた掛け算

8100H 番地 (AP 番地) の内容と 8101H 番地 (AQ 番地) の内容を掛け算し、その結果を 8102H, 8103H 番地 (AR, AR+1 番地) に格納するプログラムを作成せよ。

【ヒント】 (AP 番地) × (AQ 番地)
8 bit 8 bit

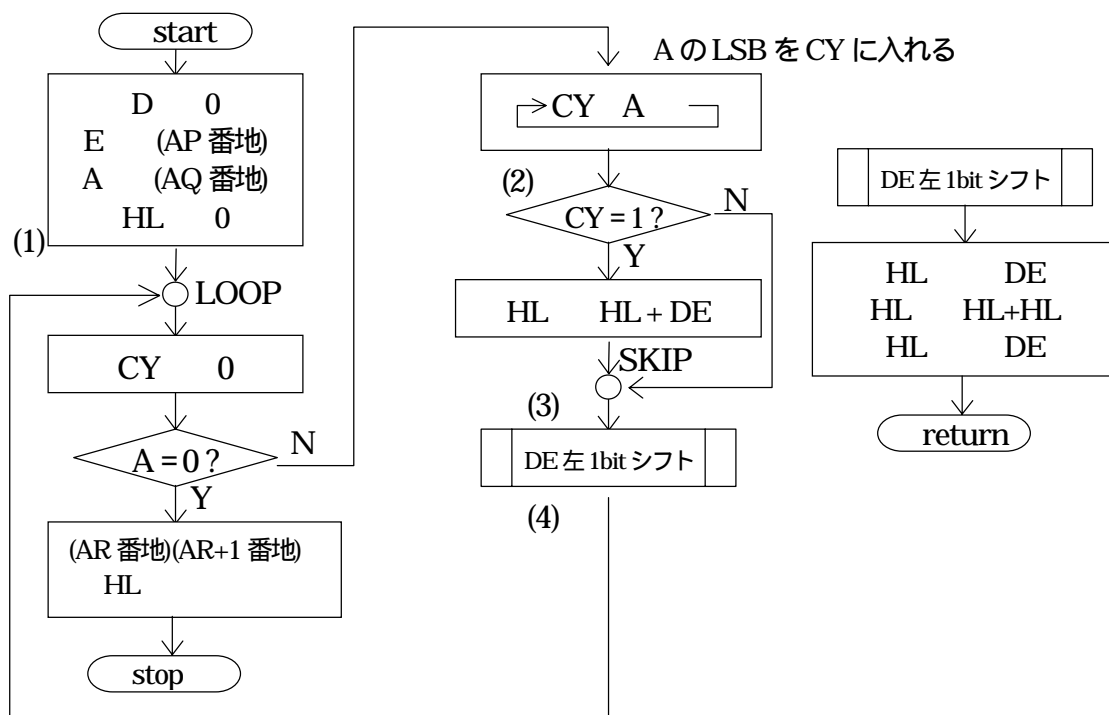
(AR 番地)(AR+1 番地)
16 bit

```

      0 1 1 1 (7)
    × 1 0 1 1 (11)
    -----
      0 1 1 1
     0 1 1 1
      0
    -----
    0 1 1 1
    1 0 0 1 1 0 1
  
```

```

DE (AP 番地)
A (AQ 番地)
HL 0, HL HL + DE
DE 左 1bit シフト, HL HL + DE
DE 左 1bit シフト
DE 左 1bit シフト, HL HL + DE
(AR 番地)(AR+1 番地) HL
  
```



	CY	A	DE	HL	
(1)		1011	0111	0	
(2)	1	101	0111	0	CY A の LSB
(3)	1	101	0111	0111	HL HL + DE
(4)		101	01110	0111	DE 左 1bit シフト
(2)	1	10	01110	0111	CY A の LSB
(3)	1	10	01110	10101	HL HL + DE
(4)		10	011100	10101	DE 左 1bit シフト
(2)	0	1	011100	10101	CY A の LSB
(3)	0	1	011100	10101	
(4)		1	0111000	10101	DE 左 1bit シフト
(2)	1	0	0111000	10101	CY A の LSB
(3)	1	0	0111000	1001101	HL HL + DE
(4)		0	01110000	1001101	DE 左 1bit シフト

5.12. サブルーチン

- ・プログラムがわかりやすくなる。
- ・同一処理を繰り返し使うとき、メモリー効率が良くなる。

【例1】 (HL) B+C

；メインルーチン

```

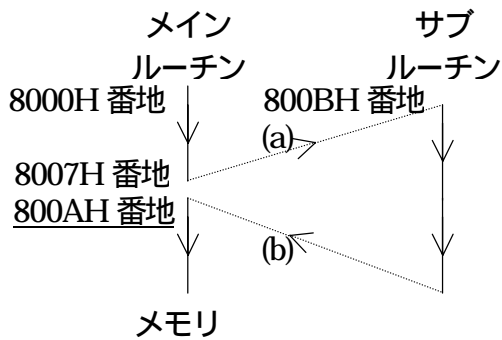
ORG 8000H
8000H   LD B,1           ; B 1
8002H   LD C,2           ; C 2
8004H   LD HL, 8100H     ; HL 8100H
8007H   CALL ADD_BC
800AH   HALT
    
```

；サブルーチン (HL) B+C

ADD_BC:

```

800BH   LD A,B           ; A B
        ADD A,C           ; A A+C
        LD (HL),A        ; (HL) A
        RET
    
```



(a) サブルーチンコール時

```

CALL 800BH
PC  PC+3
(SP - 1),(SP - 2) PC
SP  SP - 2
PC  800BH
    
```

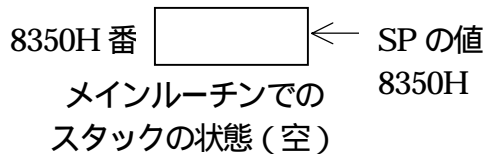
(b) リターン時

```

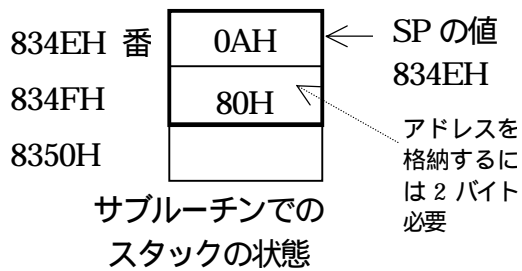
RET
PC  (SP+1),(SP)
SP  SP+2
    
```

(1) 初期状態

(3) RET 後



(2) CALL 後



【例2】かけ算サブルーチン (HL) E×A

MLT:

PUSH AF ; AF をスタックに退避

PUSH DE ; DE をスタックに退避

かけ算

; この部分で AF, DE の内容が変化してもサブルーチン終了後の
; AF, DE の内容は元の値に戻る

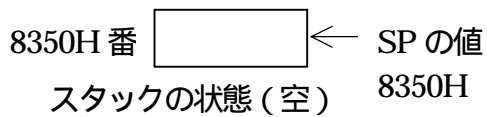
POP DE ; スタックに保存されている内容を DE に戻す

POP AF ; スタックに保存されている内容を AF に戻す

RET

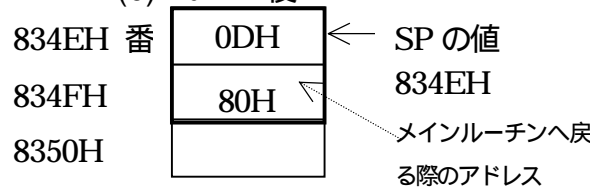
(1) 初期状態

(7) RET 後



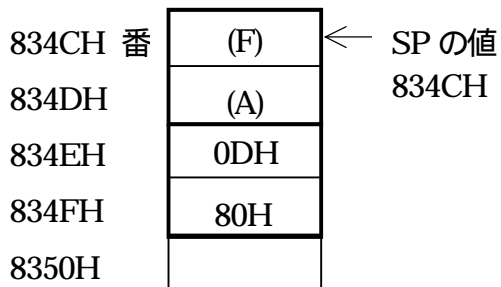
(2) CALL 後

(6) POP AF 後

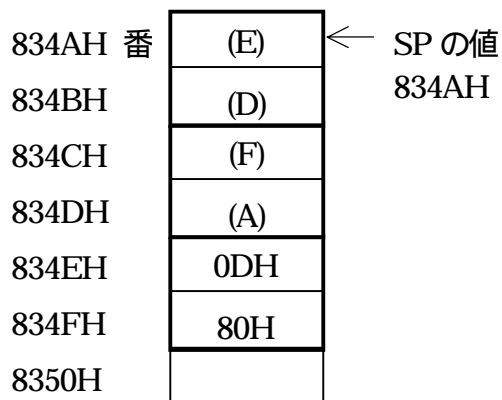


(3) PUSH AF 後

(5) POP DE 後

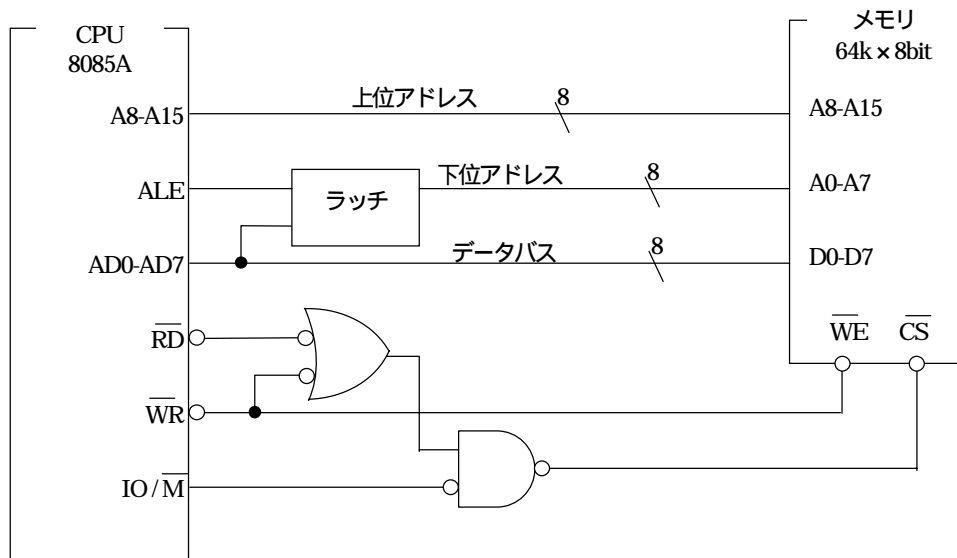


(4) PUSH DE 後



6. 周辺機器制御

(1) CPU8085A と 64k 番地 × 8bit メモリとの接続



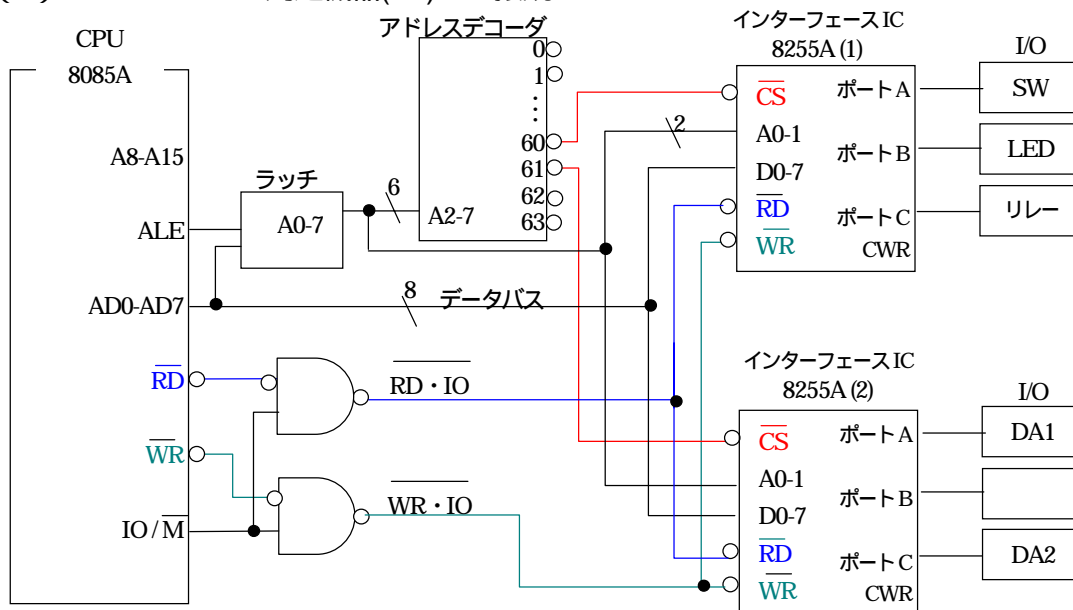
メモリ \overline{WE} ... 0:書き込み、1:読み込み

\overline{CS} ... 0を入力したとき、指定された番地のメモリとデータバスを電氣的に接続

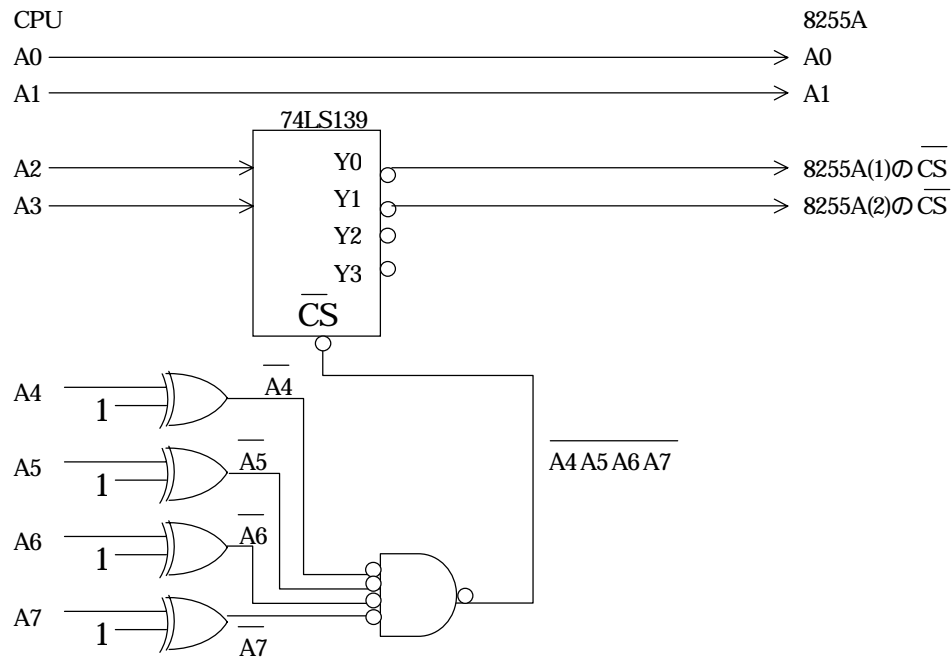
メモリ \overline{WE} CPU \overline{WR}

メモリ \overline{CS} CPU $\overline{M} \cdot (\overline{RD} + \overline{WR})$

(2) CPU8085A と周辺機器(I/O)との接続

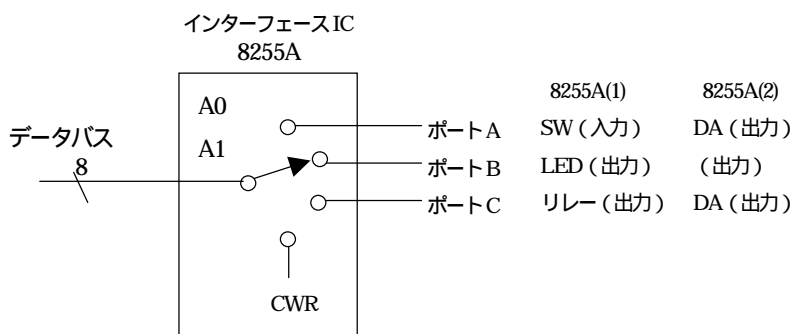


(3) アドレスデコーダ回路



		A7	A6	A5	A4	A3	A2	A1	A0	I/O アドレス	
8255A (1)	ポート A	1	1	1	1	0	0	0	0	0F0H	SW など
	ポート B	1	1	1	1	0	0	0	1	0F1H	LED
	ポート C	1	1	1	1	0	0	1	0	0F2H	リレーなど
	CWR	1	1	1	1	0	0	1	1	0F3H	コントロールワードレジスタ
8255A (2)	ポート A	1	1	1	1	0	1	0	0	0F4H	DA
	ポート B	1	1	1	1	0	1	0	1	0F5H	
	ポート C	1	1	1	1	0	1	1	0	0F6H	DA
	CWR	1	1	1	1	0	1	1	1	0F7H	コントロールワードレジスタ

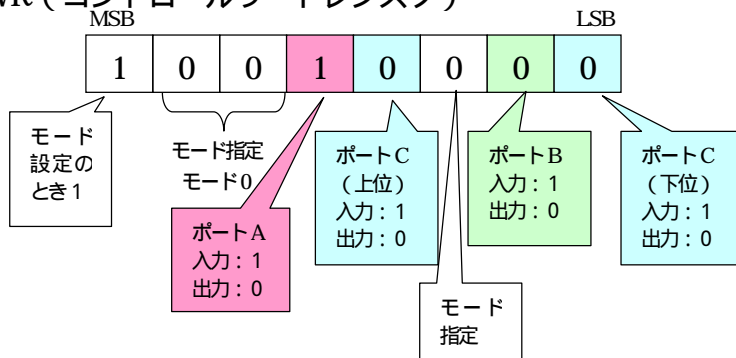
(4) インターフェース IC 8255A



CWR (コントロールワードレジスタ) ... ここに保存されているデータによって、各ポートとデータバス間のデータの流れの方向を決める。

従って、8255A を使用するには、各ポートに入力装置が接続されているのか、出力装置が接続されているのかを CWR に値をセットして指定しなければならない。

CWR (コントロールワードレジスタ)



従って、8255A(1)のCWRには90H、8255A(2)のCWRには____をセットする必要がある。

【例1】以下のようにLEDを光らせる。

○ ○ ○ ● ● ○ ○ ●

```
ORG 8000H
LD A, 90H ;A 90H
OUT (0F3H), A ;8255A(1)のCWRに90Hをセット
LD A, 80H ;A 80H
OUT (0F7H), A ;8255A(2)のCWRに80Hをセット
```

```
LD A, 00011001B ;Aに2進数で00011001を代入
OUT (0F1H), A ;(LED) A
HALT
```

【例2】例1で光らせたLEDを左にシフトさせながら表示し続ける。

時刻1 ○ ○ ○ ● ● ○ ○ ●

時刻2 ○ ○ ● ● ○ ○ ● ○

時刻3 ○ ● ● ○ ○ ● ○ ○

時刻4 ● ● ○ ○ ● ○ ○ ○

時刻5 ● ○ ○ ● ○ ○ ○ ●

時刻6 ○ ○ ● ○ ○ ○ ● ●

時間待ち
wait

【例3】押したSWに対応するLEDを点灯させる。

```
ORG 8000H
    CALL INIT    ; 初期化
LOOP:  IN A,(SW) ; A ← (SW)
    XOR 0FFH    ; A の全 bit を反転
    AND 0FH     ; A の下位 4bit 取り出し
    OUT (LED),A ; (LED) ← A
    JP LOOP     ; LOOP へ戻る

INIT:  LDA,90H
    OUT (CWR1),A
    LDA,80H
    OUT (CWR2),A
    RET

SW:    EQU    0F0H
LED:   EQU    0F1H
CWR1:  EQU    0F3H
CWR2:  EQU    0F7H
END
```

【例4】4つのSWを4bitの2進数入力とみなし、その数だけLEDに表示する値を増加させる。すなわち、sw0が押されたときLEDに表示する値を1増やし、sw1押されたときはLEDに表示する値を2増やす。ただし、LEDに表示する初期値を0とする。また、スイッチ入力されたとき、8255A(1)Aポートの下位4bitの内、対応するbitが0になる(スイッチ入力されていないときは、1になる)。さらにスイッチのチャタリングに注意すること。

```
ORG 8000H
    CALL INIT    ; 初期化
    LD C,0      ; C ← 0
LOOP:  IN A,(SW) ; A ← (SW)
    XOR 0FFH    ; A の全 bit を反転
    AND 0FH     ; A の下位 4bit 取り出し
    JP Z,LOOP   ; スイッチ入力がなければLOOPへ
    ADD A,C     ; A ← A + C
    LD C,A      ; C ← A
    OUT (LED),A ; (LED) ← A
    CALL SWOFF  ; SW が離されたことを確認
    JP LOOP     ; LOOP へ戻る

SWOFF: ; SW が離されたことを確認
    CALL DELAY ; 時間稼ぎ
    IN A,(SW)  ; A ← (SW)
    XOR 0FFH   ; A の全 bit を反転
    AND 0FH    ; A の下位 4bit 取り出し
    JP NZ,SWOFF; スイッチが押されていたらSWOFFへ
    RET
```

```

INIT:  LDA,90H
      OUT (CWR1),A
      LDA,80H
      OUT (CWR2),A
      RET

```

```

DELAY: PUSH AF
      PUSH DE
      LD DE,0681H

```

```

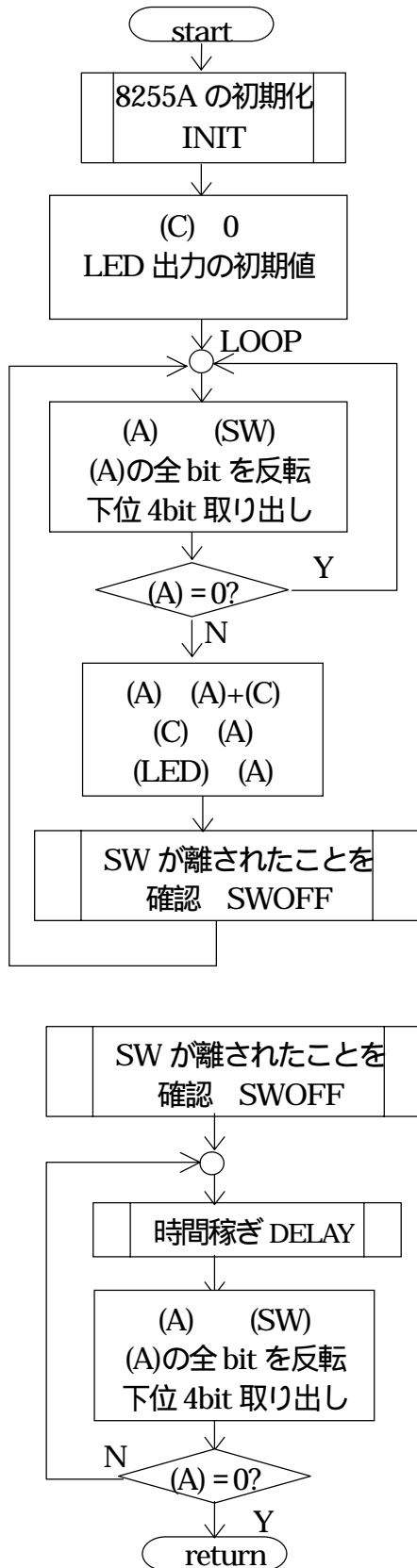
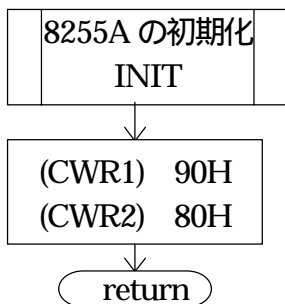
DELAY1:
      DEC DE
      LDA,D
      OR E
      JP NZ,DELAY1
      POP DE
      POP AF
      RET

```

```

SW:    EQU    0F0H
LED:   EQU    0F1H
CWR1:  EQU    0F3H
CWR2:  EQU    0F7H
END

```



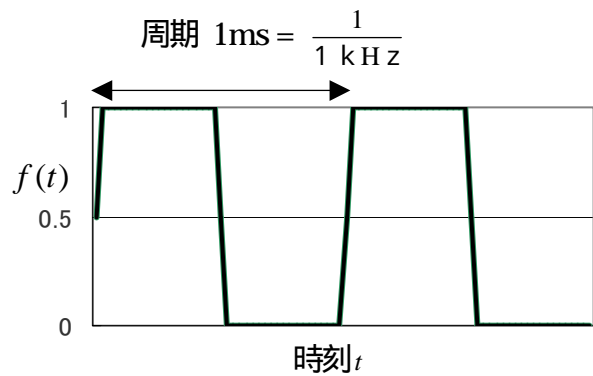
【例5】スピーカ制御

```
-----  
; スピーカ制御  
; 連続的に低い音から高い音に変化して繰り返す音  
; スピーカ接続 -- 8255A(1) C ポートのMSB  
;                ここに0,1を交互に出力すれば良い。  
;                低い音 = 周波数が低い -- 0,1の反転時間が長い  
;                高い音 = 周波数が高い -- 0,1の反転時間が短い  
; 8255A(1) C ポートのMSBのbit設定 -- CWRを使用する。  
-----  
ORG 8000H  
        CALL INIT        ; 初期化  
  
L0:     LD B,9FH         ; B ← 9FH  
        CALL SOUND       ; 低い音から高い音に変化して繰り返す音を出す  
        JP L0  
  
; 8255A(1),(2)の初期化  
INIT:   LDA,90H  
        OUT (CWR1),A  
        LDA,80H  
        OUT (CWR2),A  
        RET  
  
; 低い音から高い音に変化して繰り返す音を出す  
SOUND:  CALL SPK_ON      ; スピーカに1を出力  
        CALL SPK_OFF    ; スピーカに0を出力  
        DEC B           ; Bの値を1減らす  
        RET Z           ; Bが0になったら RETURN  
        JP SOUND  
  
; スピーカに1を出力  
SPK_ON: LDA,0FH         ; 8255A(1)のCWRに0FHを出力し、  
        OUT (CWR1),A    ; CポートのMSBをセット  
        CALL DELAY      ; Bに対応する時間待ち  
        RET  
  
; スピーカに0を出力  
SPK_OFF: LDA,0EH        ; 8255A(1)のCWRに0EHを出力し、  
        OUT (CWR1),A    ; CポートのMSBをリセット  
        CALL DELAY      ; Bに対応する時間待ち  
        RET  
  
; Bに対応する時間待ち  
DELAY:  LDA,B  
DD:     DEC A  
        JP NZ,DD  
        RET  
  
CWR1:   EQU    0F3H  
CWR2:   EQU    0F7H  
END
```

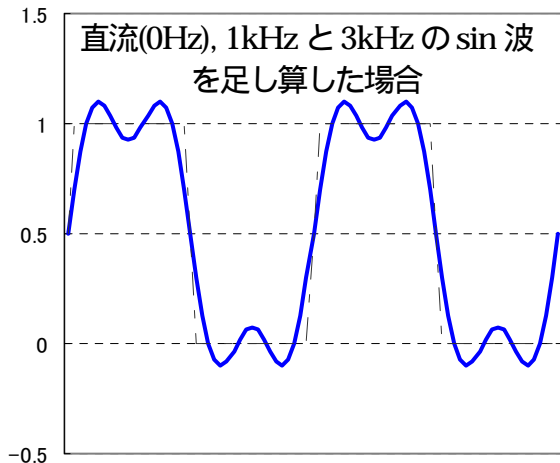
7. AD変換, DA変換

7.1. スペクトルとは

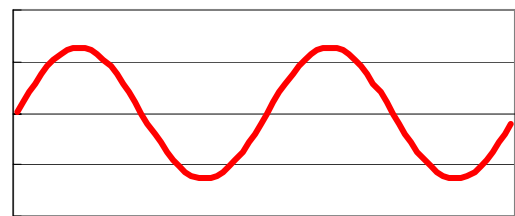
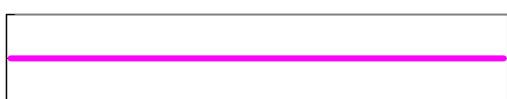
図(a)は、1kHzの方形波である。
この方形波は、(b)~(e)の波形の和で
近似できる。



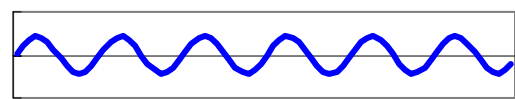
(a) 方形波



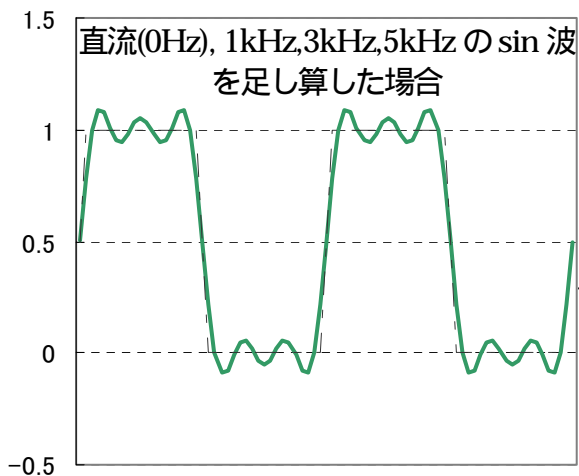
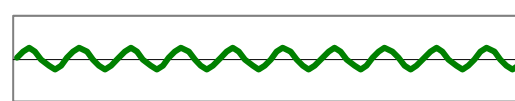
(b) $\frac{1}{2}$ (直流 0Hz)



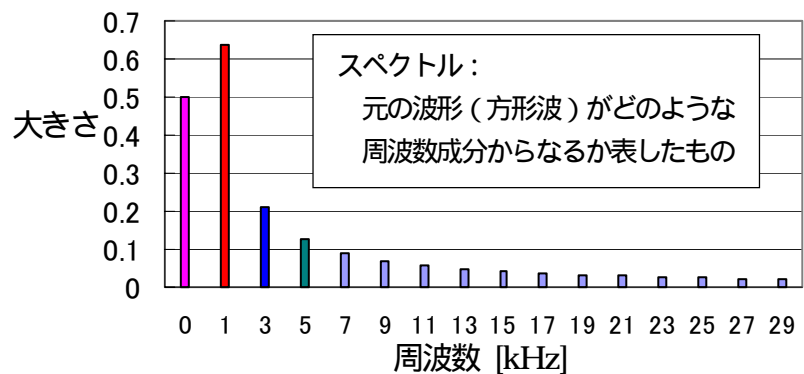
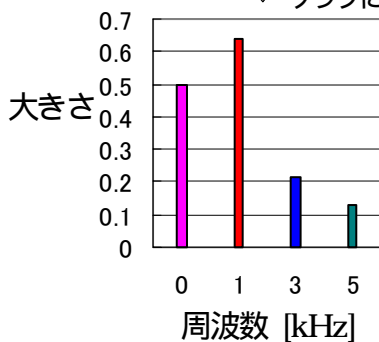
(d) 3kHz の sin 波 $\frac{2}{\pi} \frac{1}{3} \sin(2\pi 3 f_0 t)$



(e) 5kHz の sin 波 $\frac{2}{\pi} \frac{1}{5} \sin(2\pi 5 f_0 t)$

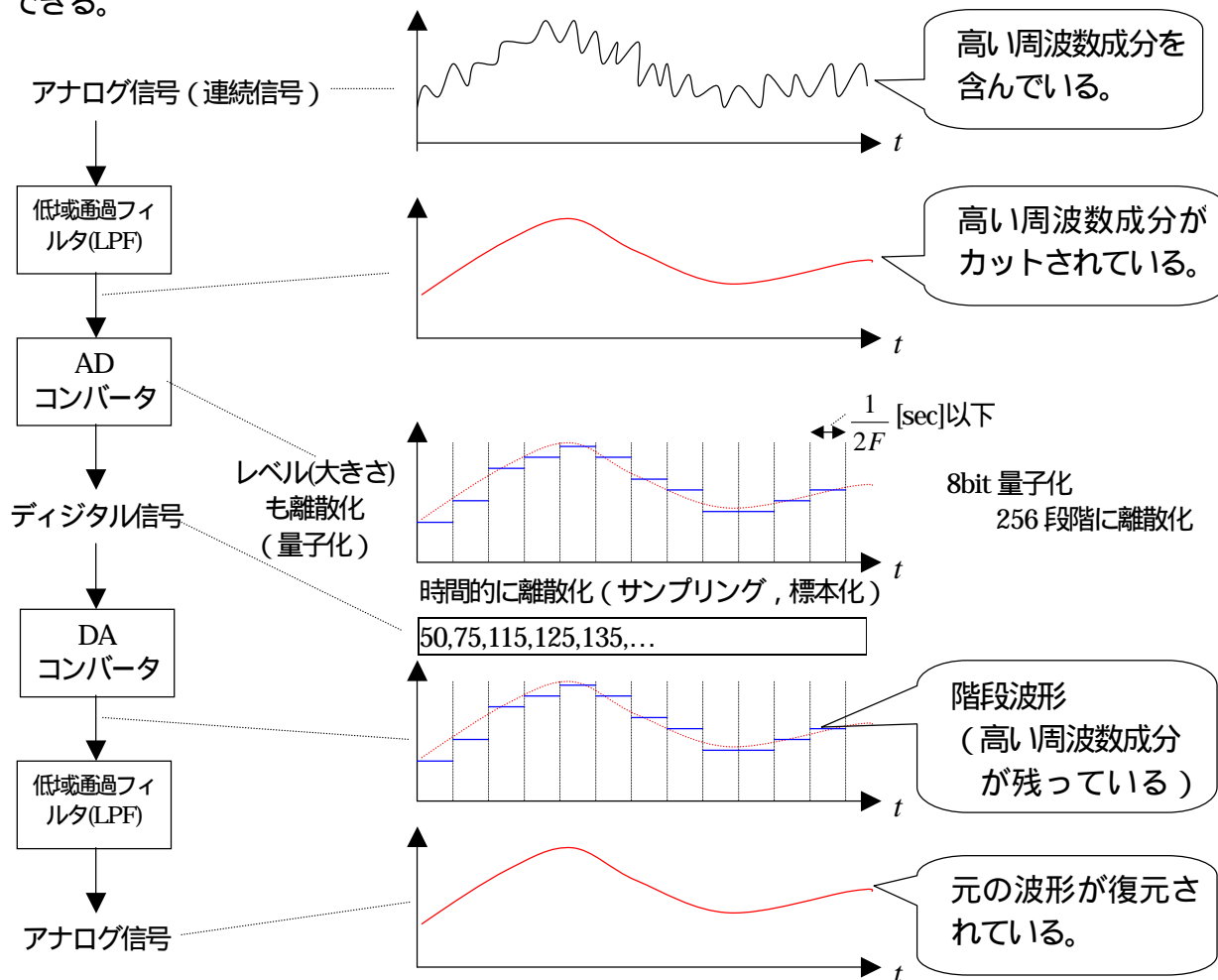


足し算した直流(0Hz), 1kHz,
3kHz, 5kHz の成分の大きさを
グラフにすると...



7.2. サンプリング定理 (標本化定理)

信号の帯域が F [Hz]以下であるとき (F [Hz]より大きい周波数成分を持たないとき) $\frac{1}{2F}$ [sec]周期以下 ($2F$ [Hz]以上) でサンプリング (標本化) すれば、完全に元の信号を復元できる。

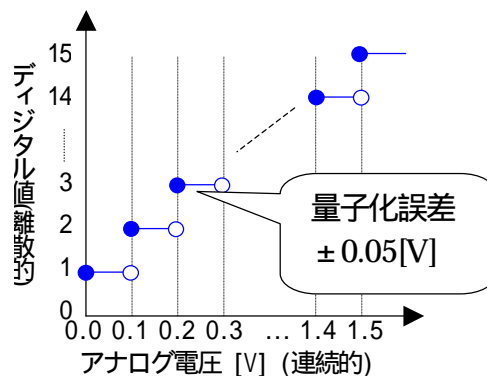


7.3. DA 変換 (デジタル信号 アナログ信号)

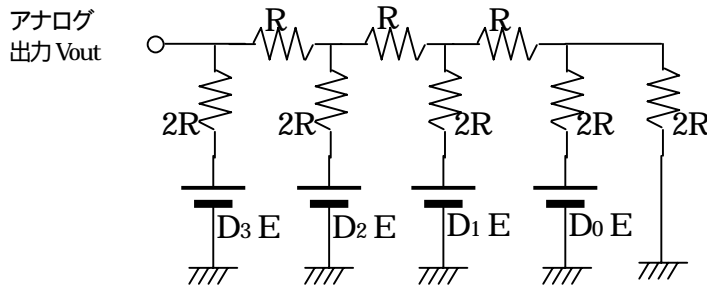
【例】4 bit でレベル (大きさ) を表現した(4 bit 量子化された)デジタル信号の場合

デジタル値とアナログ電圧の対応例

10進数	デジタル値				アナログ電圧 [V]
	D3	D2	D1	D0	
0	0	0	0	0	0.0
1	0	0	0	1	0.1
2	0	0	1	0	0.2
...
15	1	1	1	1	1.5



【R - 2R はしご型 DA コンバータ回路例】



アナログ出力

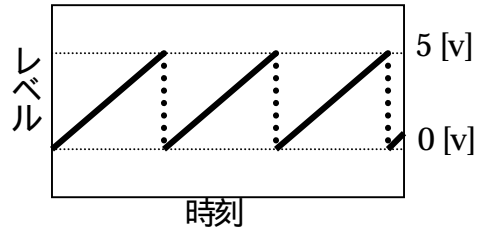
$$V_{out} = \frac{E}{2^4} (2^3 D_3 + 2^2 D_2 + 2^1 D_1 + D_0)$$

E = 1.6[V] とすれば左上の表の
ような対応になる。

【のこぎり波の表示】

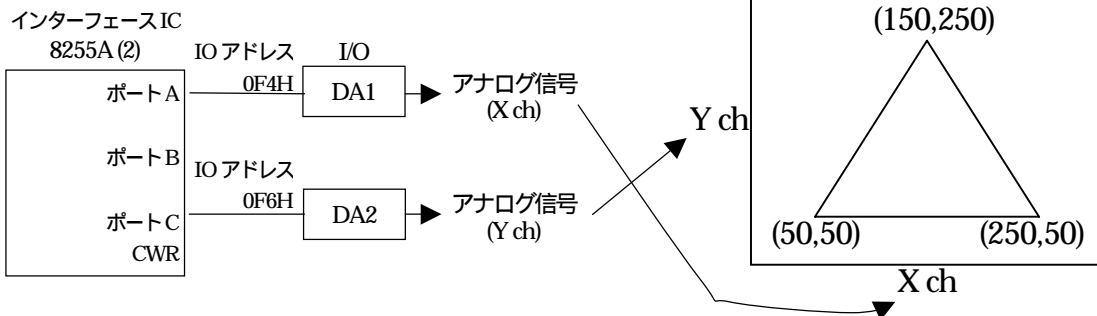
オシロスコープに右図に示すのこぎり波が表示される Z80 アセンブリプログラムを作成してみよう。

ただし、IO アドレス 0F4H は 8255A(2) Aポートに対応し、8bitDA コンバータが接続されている。また8bitDA コンバータ出力は、オシロスコープにあらかじめ接続してあるものとする。なお、この8bitDA コンバータでは、デジタル値 0H が 0 [V]、デジタル値 0FFH が 5 [V] のアナログ電圧に対応するものとする。

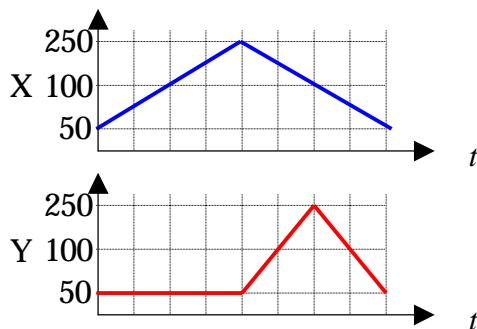


オシロスコープへの出力波形

【三角形の表示】



オシロスコープに上図に示す三角形が表示される Z80 アセンブリプログラムを作成してみよう。
(ヒント)



7.4. AD 変換 (アナログ信号 デジタル信号)

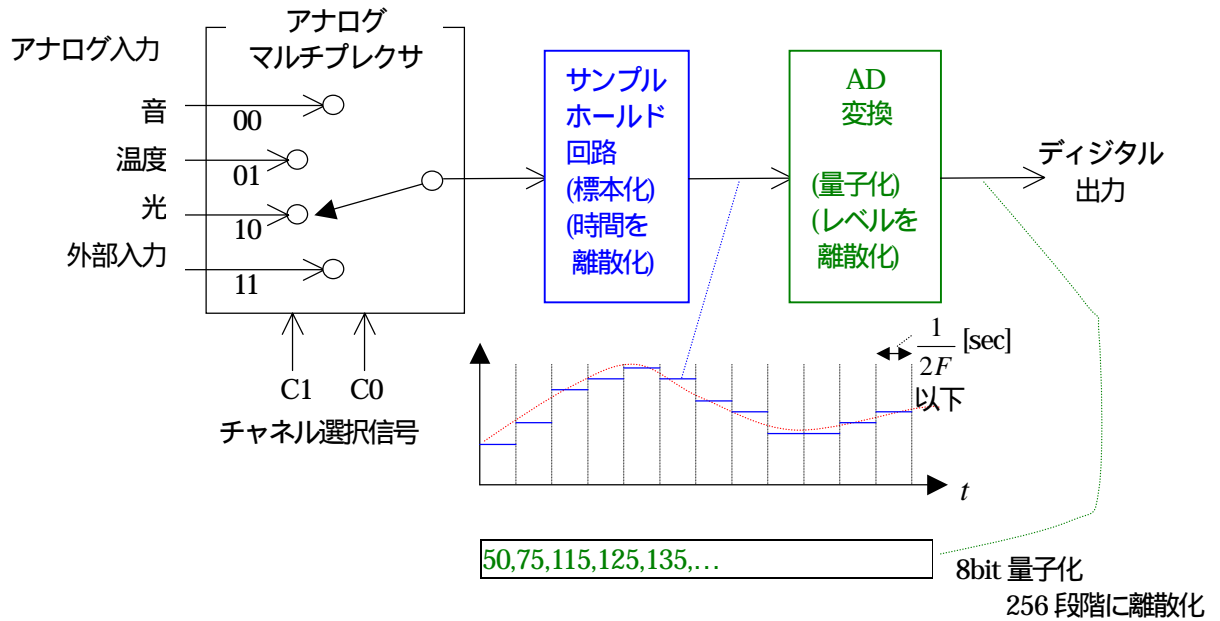
(1) AD コンバータの種類

低速用...カウンタランプ型, 2重積分型

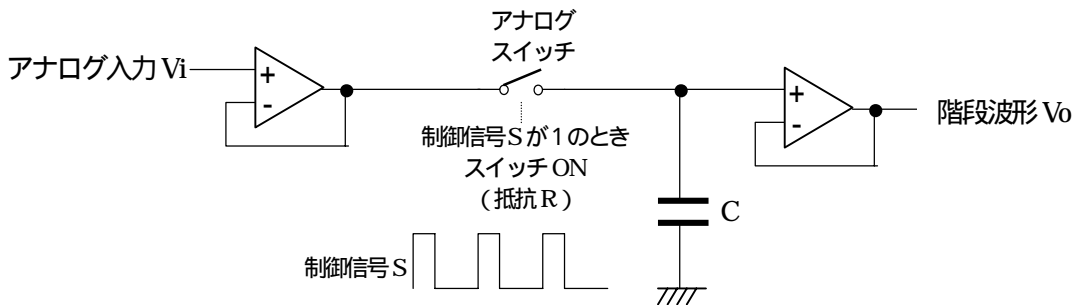
中速用...逐次比較型

高速用...並列比較型

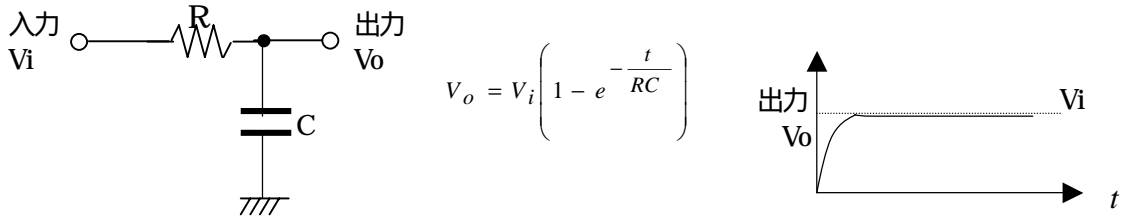
(2) AD コンバータの構成



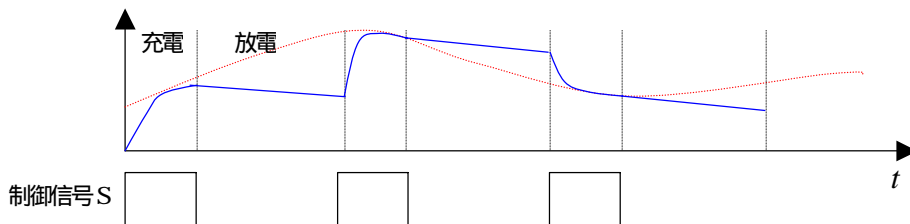
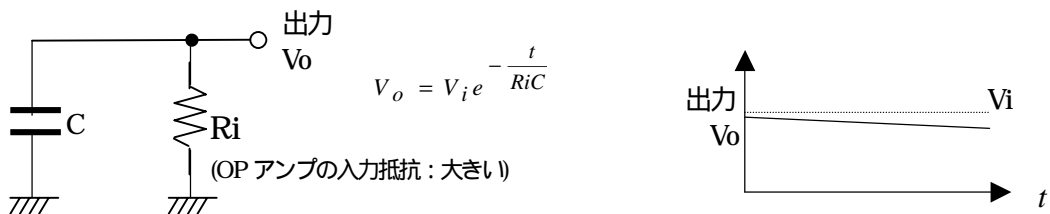
(3) サンプルホールド回路例



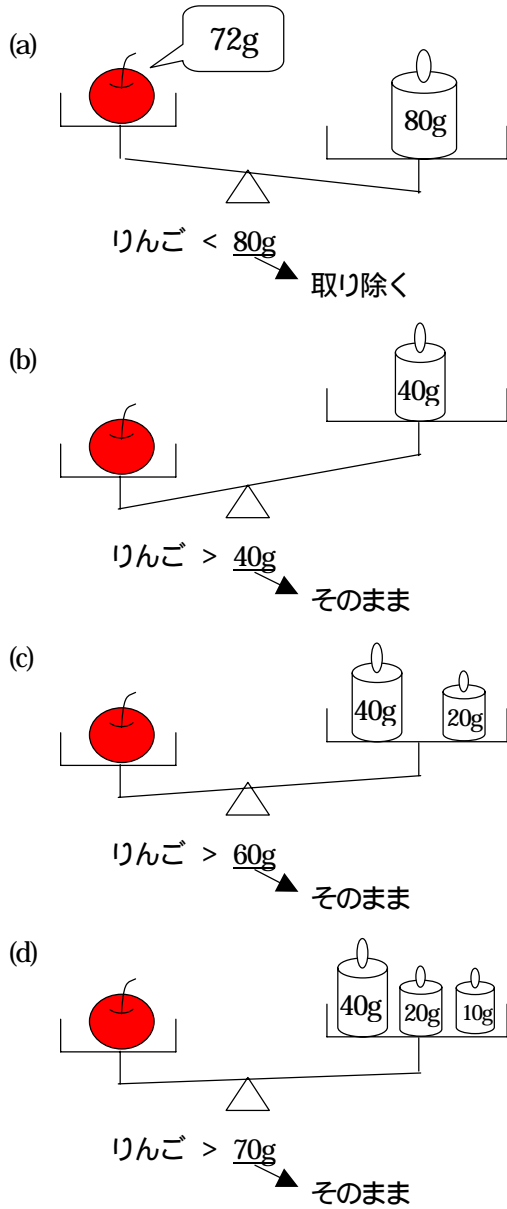
【S=1 スイッチ ON 急速充電】



【S=0 スイッチ OFF ゆっくり放電】



(4) 量子化 (逐次比較型)
【りんごの重さを量るとき】



【アナログ入力電圧を測るとき】

10進数	デジタル値				アナログ電圧 [V]
	D3	D2	D1	D0	
0	0	0	0	0	0.0
1	0	0	0	1	0.1
2	0	0	1	0	0.2
15	1	1	1	1	1.5

(a) アナログ入力電圧 < 基準電圧 0.8[V]
0.72[V] → 1 0 0 0 (D3のみ1) → 取り除く

(b) アナログ入力電圧 > 基準電圧 0.4[V]
0 1 0 0 (D2のみ1) → そのまま

(c) アナログ入力電圧 > 基準電圧 0.6[V]
0 1 1 0 (D1を1に) → そのまま

(d) アナログ入力電圧 > 基準電圧 0.7[V]
0 1 1 1 (D0を1に) → そのまま

アナログ入力電圧(0.72[V])
0 1 1 1の4bit デジタル値に変換

【4bit 逐次比較型 AD 変換回路例】

アナログ入力電圧 $V_i=0.72[V]$ のとき

(a) クロック

リングカウンタ Q1 Q2 Q3 Q4 Q5
1 0 0 0 0

($Q_1 = 1$) FF3 の Set 端子

DA コンバータ入力: 1 0 0 0

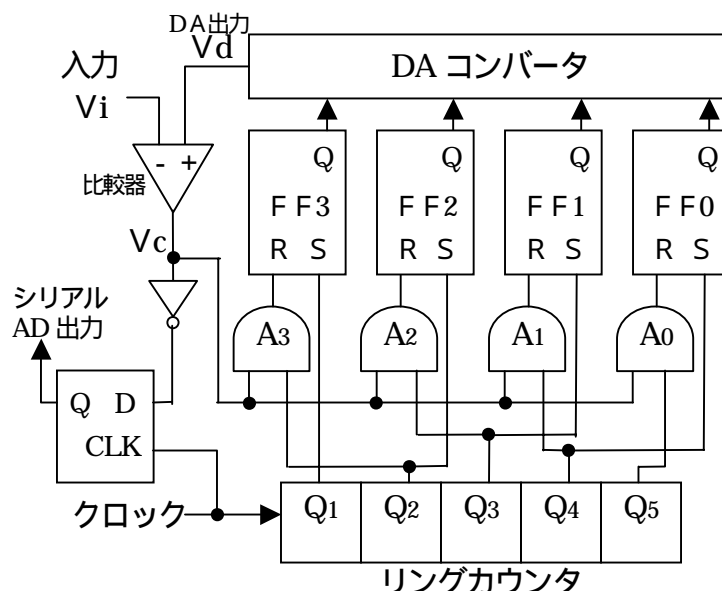
DA 出力 (基準電圧 V_d): 0.8[V]

アナログ入力電圧 $V_i >$ 基準電圧 V_d なら

比較器出力 $V_c = 0$ (おもりはそのまま)

アナログ入力電圧 $V_i <$ 基準電圧 V_d なら

比較器出力 $V_c = 1$ (おもりを取り除く)



(b) クロック 前段の結果($D_3=0$)がシリアル出力される

リングカウンタ Q1 Q2 Q3 Q4 Q5 ($Q_2 = 1$)
0 1 0 0 0

$Q_2 = 1$, $V_c = 1$ が A3 に入力 FF3 の Reset 端子に 1 入力 FF3 出力: 0

$Q_2 = 1$ が FF2 の Set 端子に入力 FF2 出力: 1

DA コンバータ入力: 0 1 0 0 DA 出力 (基準電圧 V_d): 0.4[V]

アナログ入力電圧 $V_i >$ 基準電圧 V_d なので、比較器出力 $V_c = 0$ (おもりはそのまま)

(c) クロック 前段の結果($D_2=1$)がシリアル出力される

リングカウンタ Q1 Q2 Q3 Q4 Q5 ($Q_3 = 1$)
0 0 1 0 0

$Q_3 = 1$, $V_c = 0$ が A3 に入力 FF2 の Reset 端子に 0 入力 FF2 出力: 1 のまま

$Q_3 = 1$ が FF1 の Set 端子に入力 FF1 出力: 1

DA コンバータ入力: 0 1 1 0 DA 出力 (基準電圧 V_d): 0.6[V]

アナログ入力電圧 $V_i >$ 基準電圧 V_d なので、比較器出力 $V_c = 0$ (おもりはそのまま)

(d) クロック 前段の結果($D_1=1$)がシリアル出力される

リングカウンタ Q1 Q2 Q3 Q4 Q5 ($Q_4 = 1$)
0 0 0 1 0

$Q_4 = 1$, $V_c = 0$ が A3 に入力 FF1 の Reset 端子に 0 入力 FF1 出力: 1 のまま

$Q_4 = 1$ が FF0 の Set 端子に入力 FF0 出力: 1

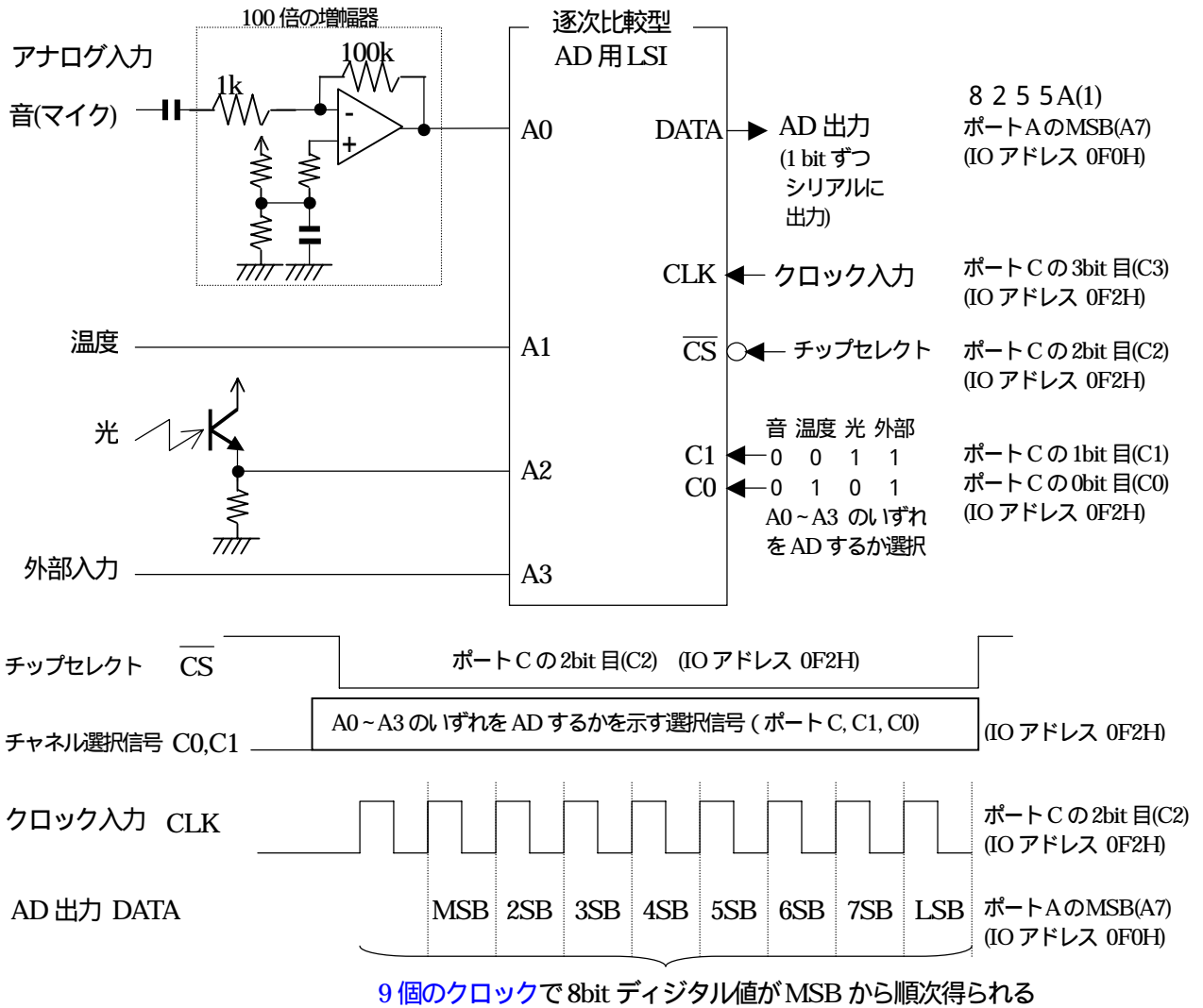
DA コンバータ入力: 0 1 1 1 DA 出力 (基準電圧 V_d): 0.7[V]

アナログ入力電圧 $V_i >$ 基準電圧 V_d なので、比較器出力 $V_c = 0$ (おもりはそのまま)

(e) クロック 前段の結果($D_0=1$)がシリアル出力される

以上のように、入力電圧と基準電圧を逐次比較することで AD 変換を行える。

(5) IOボードのAD変換部



(6) AD変換プログラム例

```

*****
;
; AD 変換
; 音(0)、温度(1)、光(2)、外部入力(4)
; のいずれかを AD 変換し、結果を LED に
; 表示する。
*****
; メインルーチン
ORG 8000H ; 8000H 番地から
LD A,90H ; 1 つめの 8255A を
OUT (CMR1),A ; A:入力、B,C:出力に初期化
LD A,80H ; 2 つめの 8255A を
OUT (CMR2),A ; A,B,C:出力に初期化

LD A,4 ; _
OUT (PORTC1),A ; CS <-- High

LD C,SELECT ; 入力を選択 音(0)、温度(1)、光(2)、外部入力(4)
LOOP: CALL ADC ; AD 変換 (結果はAレジスタに格納)
OUT (PORTB1),A ; AD 変換結果を LED に出力
JP LOOP

```

```

; AD 変換サブルーチン
; 入力: C レジスタ 音(0)、温度(1)、光(2)、外部入力(3)
; 出力: A レジスタ AD 変換結果

```

```

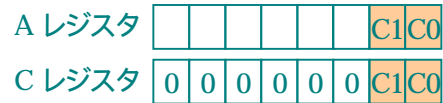
ORG 8100H ; 8100H 番地から
ADC: PUSH BC ; BC を退避 (サブルーチン終了後、元に戻すため)
    PUSH DE ; DE を退避

```

```

LD A,C ; C の下位2bit のみ
AND 3 ; を取り出す
LD C,A ;

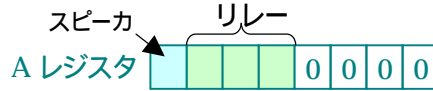
```



```

IN A,(PORTC1) ; PORTC1 の
AND OF0H ; 上位4bit を取り出し

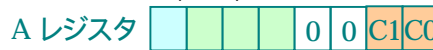
```



```

OR C ; 下位2bit(C0,C1)に選択されたAD変換元(0~4)をセット

```



```

XOR 4 ; CS --> High (反転)

```

```

OUT (PORTC1),A ;

```



```

XOR 4 ; CS --> Low (反転)

```

```

OUT (PORTC1),A ;

```



```

LD C,9 ; 9 回ループ

```

```

LD DE,0706H ; D<--7 E<-- 6

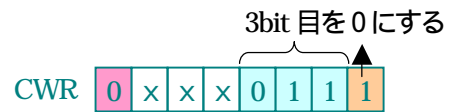
```



```

ADC1: LD A,D ;
... OUT (CWR1),A ; Clock --> High

```



```

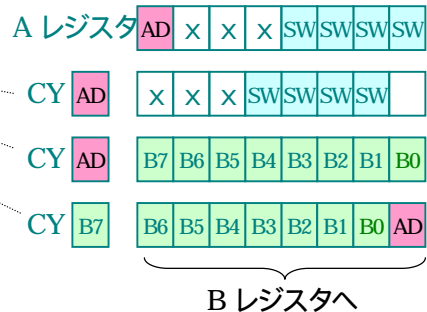
LD A,E ;
OUT (CWR1),A ; Clock --> Low

```

```

... IN A,(PORTA1) ; A のMSB:A7 <-- AD 結果
    RLA ; CY <-- A7
    LD A,B ; A <-- B
    RLA ; AO <-- CY
    LD B,A ; B <-- A

```



```

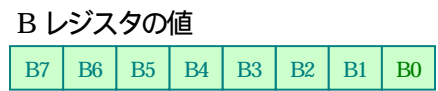
DEC C ; C --
JP NZ,ADC1 ; 9 回ループ

```

```

LD A,5 ;
OUT (CWR1),A ; CS --> High

```



```

LD A,B ; A <-- B AD 結果
POP DE ; DE を元に戻す
POP BC ; BC を元に戻す
RET

```



```

PORTA1: EQU 0F0H ; MSB: AD 結果
PORTB1: EQU 0F1H ; LED
PORTC1: EQU 0F2H ; C0,C1: AD 入力選択 C2: CS C3: Clock
CWR1: EQU 0F3H ; 1 つめの 8255A のコントロールワードレジスタ
CWR2: EQU 0F7H ; 2 つめの 8255A のコントロールワードレジスタ
SELECT: EQU 2 ; 音(0)、温度(1)、光(2)、外部入力(3)

```

```

END ; プログラム記述終了

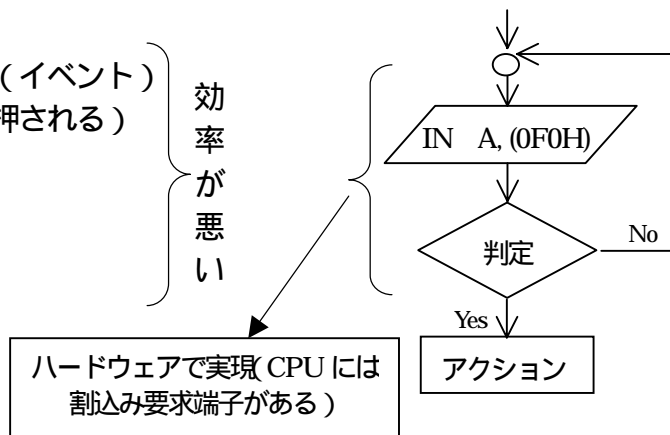
```

7.5. 入出力割り込み

(1) 入出力割り込みとは
いつ起こるか分からない事象 (イベント)
(例: 自動ドア、スイッチが押される)

CPU で常時監視

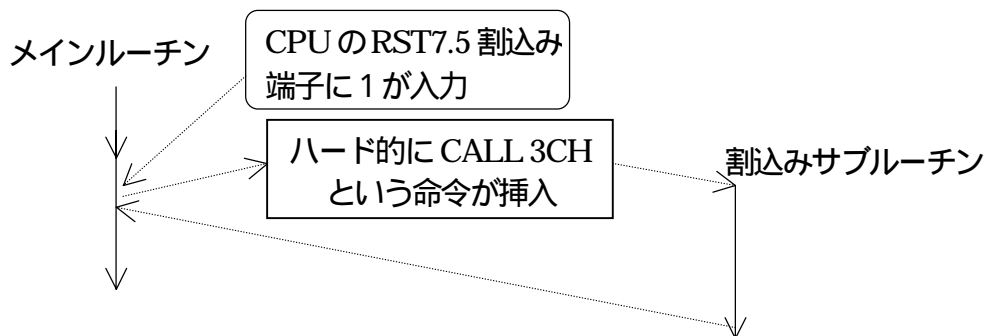
アクション



入出力割り込み処理:

CPU には別の仕事をさせておき、ハード的に事象が起こったことを検出し処理する。

【8085A RST7.5 割り込みの場合】



(2) 入出力割り込みの種類

{ ノンマスクابل割り込み ... いつでも割り込み可 (割り込みを禁止できない)
 { マスクابل割り込み ... 割り込みを禁止することができる。

{ リスタート割り込み ... 割り込みが発生したら特定番地から再開
 { ベクトル割り込み ... 割り込み発生後、選択用データ (ベクトル) を周辺機器から送り、
 どのような処理をさせるかを指示

(3) CPU 8085A の割り込み

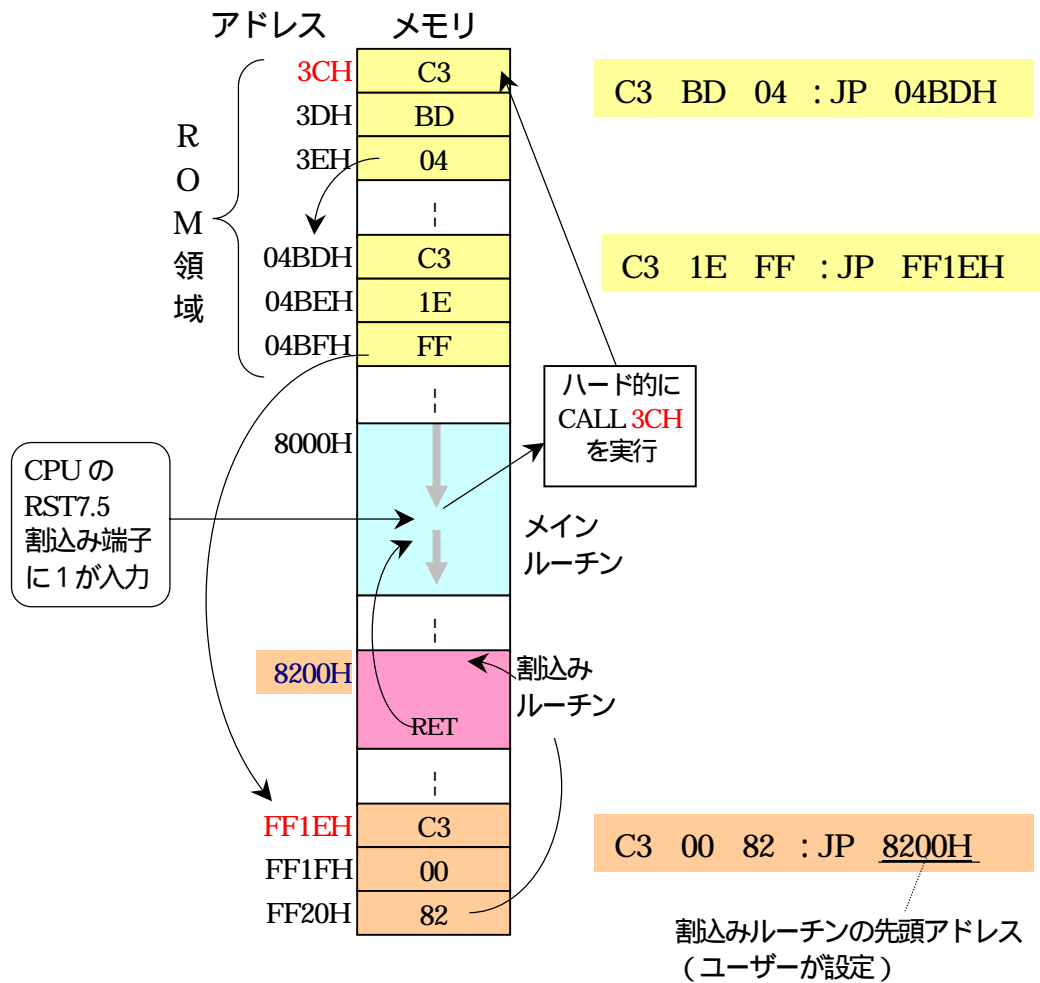
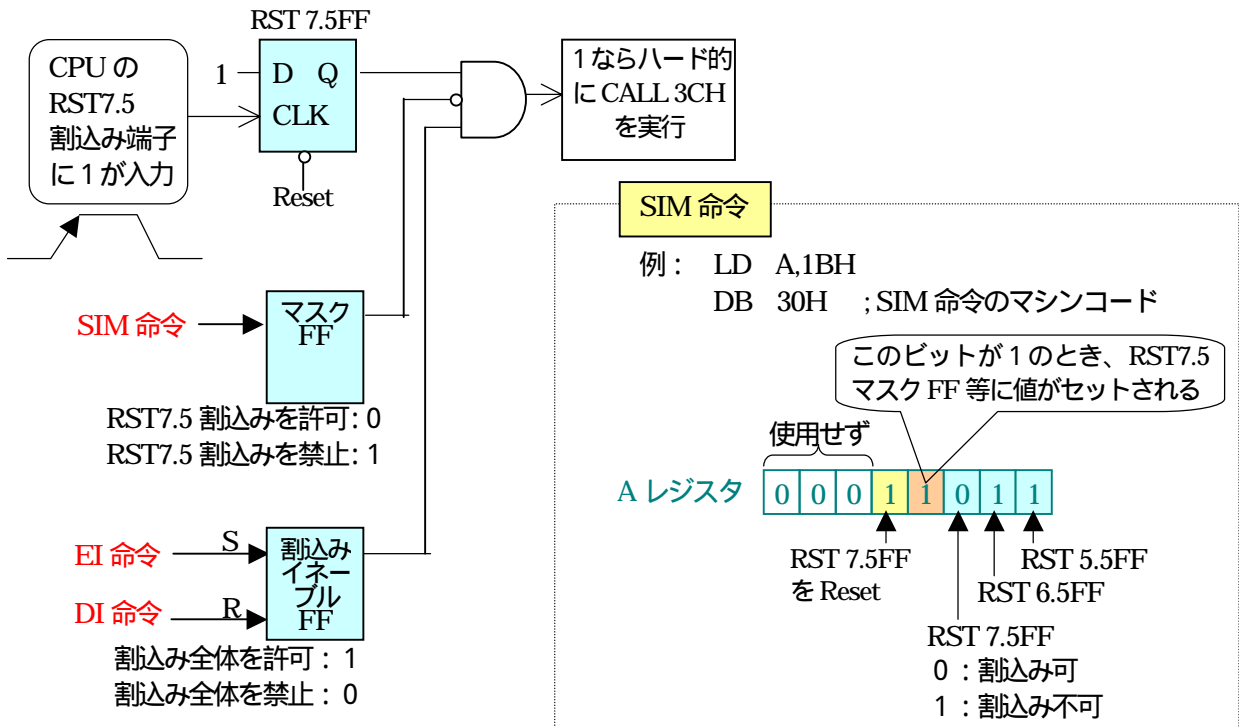
5 つの割り込み要求端子		特定番地	
RST5.5	CALL	2CH	} リスタート割り込み, マスクابل割り込み
RST6.5	CALL	34H	
RST7.5	CALL	3CH	
TRAP	CALL	24H	... リスタート割り込み, ノンマスクابل割り込み
INTR			... ベクトル割り込み, マスクابل割り込み

【INTR】

INTR 端子に 1 (High) が入力 $\overline{\text{INTA}}$ (割り込み許可端子) に 0 が出力
外部からデータバスに命令 (ベクトル) を入れる

選択用データ: RST 1, RST2, RST3, RST4, RST 5, RST 6, RST7 (1 バイト命令)

【RST7.5】



(4) RST7.5 割り込みプログラム例

```

;*****
;* 割り込みを使用した例 *
;* 通常は音をならしておき、割り込み *
;* 発生時(割り込みボタンが押されたら) *
;* リレーを2回ON/OFFさせる。 *
;*****

```

```

; メインルーチン

```

```

        ORG 8000H          ; 8000H 番地から
; 初期設定
        LD A,90H          ; 1 つめの 8255A を
        OUT (CMR1),A      ; A:入力、B,C:出力に初期化
        LD A,80H          ; 2 つめの 8255A を
        OUT (CMR2),A      ; A,B,C:出力に初期化

        LD A,4            ; —
        OUT (PORTC1),A    ; AD CS <-- High

```

```

LD A,0C3H;
LD (OFF1EH),A ; (OFF1EH 番地) <-- 0C3H (JP 命令) ;
LD HL,INT; ; JP INT
LD (OFF1FH),HL ; (OFF1FH, OFF20H 番地) <-- INT ;

```

割り込みルーチンの
先頭アドレスを設定

RST7.5
割り込みを
許可

```

LD A,1BH ; 割り込みボタンに接続されている RST7.5 割り込み
DB 30H ; 端子の割り込みが可能になるように割り込みマスクをセット(SIM 命令)

```

```

EI ; 割り込み全体を許可

```

```

; 通常ルーチン (音を鳴らす)

```

```

SOUND: LD A,0FH          ; スピーカが接続されているビット(Port C MSB)
        CALL OUTPC7      ; を High にする
        LD A,0EH          ; スピーカが接続されているビット(Port C MSB)
        CALL OUTPC7      ; を Low にする
        JP SOUND

```

```

OUTPC7: OUT (CMR1),A     ; スピーカが接続されているビットを 1/0 にする
        LD A,9FH          ; 時間稼ぎ

```

```

DELAY: DEC A
        JP NZ,DELAY
        RET

```

```

; 割り込みルーチン

```

```

; リレーを2回ON/OFFさせる

```

```

INT:   PUSH AF          ; AF を退避 (サブルーチン終了後、元に戻すため)
        PUSH HL         ; HL を退避

```

```

        LD A,09H        ; リレー 1 (Port C 4bit 目)
        CALL INT5       ; を ON にする。
        LD A,08H        ; リレー 1 (Port C 4bit 目)
        CALL INT5       ; を OFF にする。
        LD A,09H        ; リレー 1 (Port C 4bit 目)

```

```
CALL INT5      ; を ON にする。
LD A,08H      ; リレー 1 (Port C 4bit 目)
CALL INT5      ; を OFF にする。
```

```
LD A,10H      ; 割り込みボタンに接続されている RST7.5 割り込み FF
DB 30H        ; をリセットする(SIM 命令)
```

```
POP HL        ; HL を元に戻す
POP AF        ; AF を元に戻す
```

```
EI           ; 割り込み解除
```

```
RET
```

```
INT5:  OUT (CWR1),A    ; リレー1 が接続されているビットを 1/0 にする。
        LD HL,0C000H   ;
INT6:  DEC HL          ; 時間稼ぎ(wait)
        LD A,H         ;
        OR L           ;
        JP NZ,INT6     ;
        RET
```

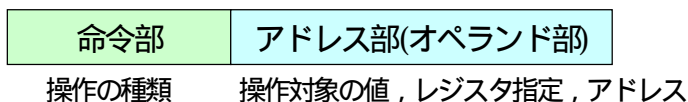
```
PORTC1: EQU 0F2H      ; C7: スピーカ  C4: リレー
CWR1:   EQU 0F3H      ; 1 つめの 8255A のコントロールワードレジスタ
CWR2:   EQU 0F7H      ; 2 つめの 8255A のコントロールワードレジスタ
```

```
END          ;プログラム記述終了
```

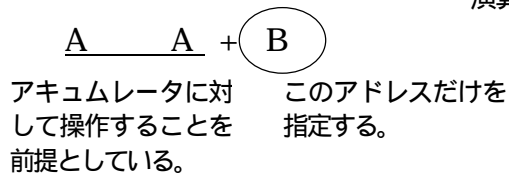
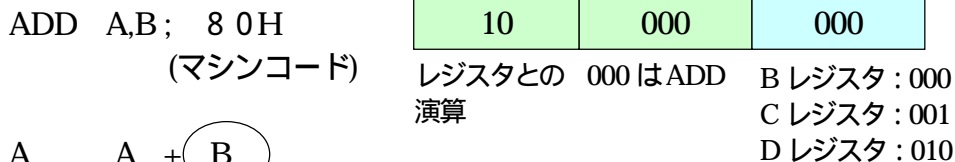
8. コンピュータアーキテクチャ

8.1. アドレス部 (オペランド部)

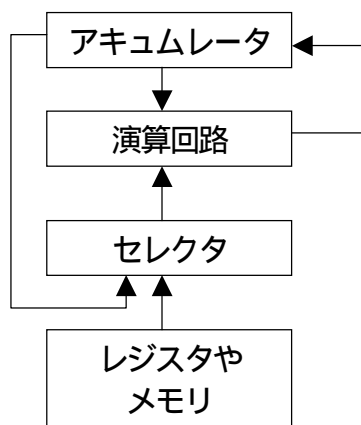
(1) 機械語命令



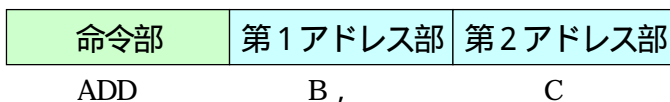
(2) 1オペランド命令 (1アドレス命令) ...アドレス部に1つのアドレスを指定



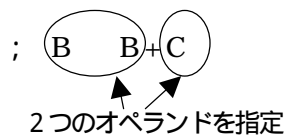
アキュムレータ型
計算機



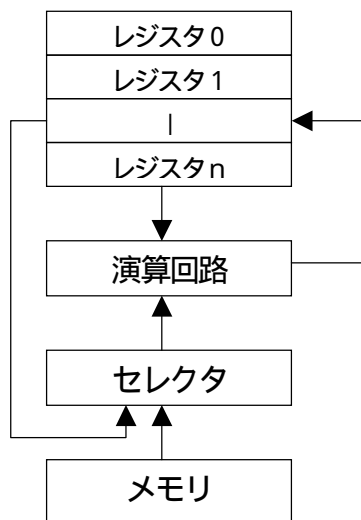
(3) 2オペランド命令 (2アドレス命令) ...アドレス部に2つのアドレスを指定



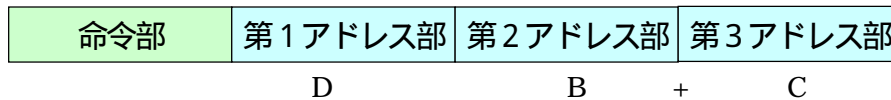
【例】LD B, C (マシンコード 4 1 H)
LD r, r'
01 000 001



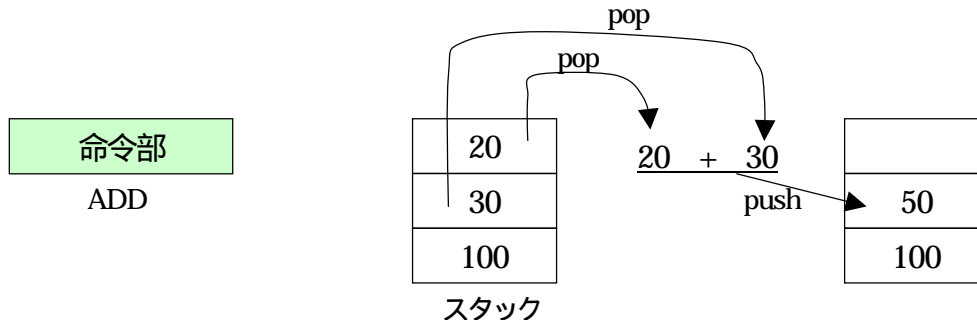
レジスタ型計算機



- (4) 3オペランド命令(3アドレス命令)
 ...アドレス部(オペランド部)に2つのアドレス(オペランド)を指定



- (5) 0アドレス計算機(スタックマシン)...アドレス指定なし



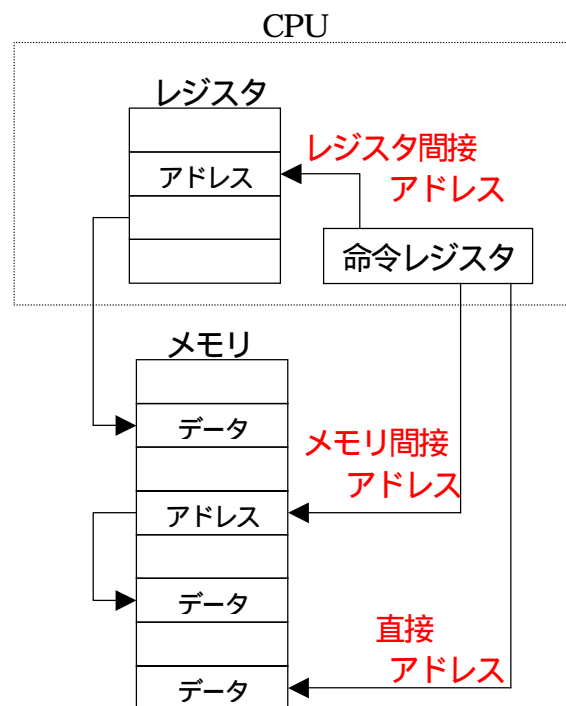
8.2. アドレス修飾

有効アドレス(実効アドレス)...命令の操作対象のデータが格納されているメモリ上の番地
 アドレス修飾...有効アドレスを指定するためのさまざまな規則

- (1) **直接**アドレス指定方式
 命令語のアドレス部で、直接有効アドレスを指定する方式
 【例】LD A, (1234H)
 (命令語のアドレス部) 有効アドレス

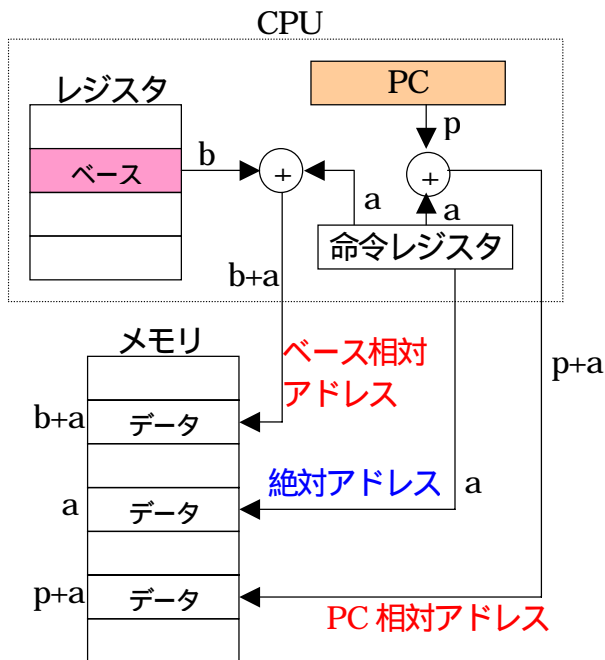
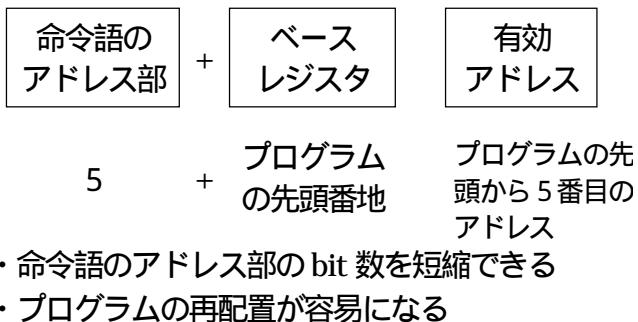
- (2) **間接**アドレス指定方式
 命令語のアドレス部で指定されたレジスタまたはメインメモリに有効アドレスが格納されており、レジスタまたはメインメモリを介して間接的に有効アドレスを指定する方式 ([3] p.49 図 2.7 参照)
 【例】LD A, (HL)
 ・レジスタ等を経由することでアドレスを指定するための bit 数を節約できる
 ・アドレスを順次変更しながら柔軟な処理を行える
 (XX 番地から 100 個のデータを YY 番地にコピーする例 p.27 5.6 (2)参照)

- (命令語のアドレス部)
 (レジスタまたはメインメモリ) 有効アドレス



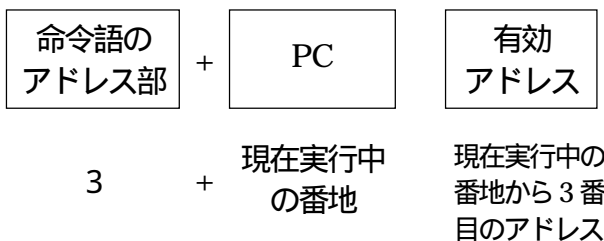
(3) **基底 (ベース) アドレス指定方式**

命令語のアドレス部の値にベースレジスタの値を加算したものを有効アドレスとする方式 ([3] p.49 図 2.8 参照)



(4) **PC 相対アドレス指定方式**

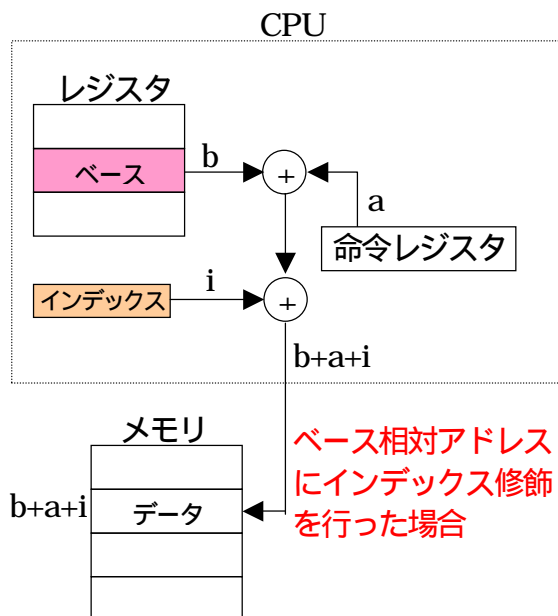
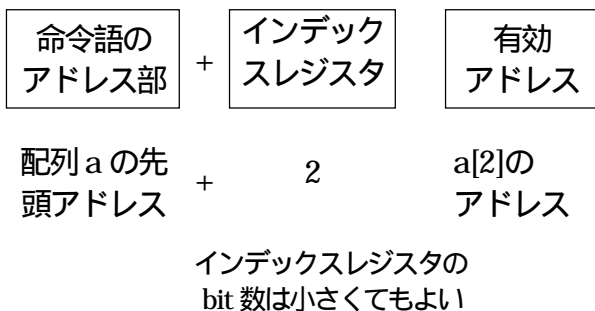
命令語のアドレス部の値に PC の値を加算したものを有効アドレスとする方式



- 命令語のアドレス部の bit 数を短縮できる
- プログラムの再配置が容易になる

(5) **指標 (インデックス) アドレス指定方式**

命令語のアドレス部の値にインデックスレジスタの値を加算したものを有効アドレスとする方式



- (6) ビッグエンディアン (big endian) とリトルエンディアン (little endian)
 2つ以上のアドレスにまたがってデータを格納する際に、
 データの下位の桁を大きいアドレスに割り振る場合 ビッグエンディアン
 データの下位の桁を小さいアドレスに割り振る場合 リトルエンディアン

【例】・リトルエンディアン(windows PC など)

データ 1234H 600 番地 : 34H (下位)

601 番地 : 12H (上位)

・ビッグエンディアン(Mac, Sun ワークステーションなど)

データ 1234H 600 番地 : 12H (上位)

601 番地 : 34H (下位)

8.3. CPUの基本動作と命令パイプライン ([1] p.58~p.60)

- (1) 命令実行手順

命令サイクル

F : 命令のフェッチ (命令の取り出し)

D : 命令のデコード (命令の解読)

E : 有効アドレスの計算と命令の実行

WB : 演算結果をメインメモリに書き戻す

- (2) 逐次制御方式

時刻 (クロック)	1	2	3	4	5	6	7	8
命令1	F	D	E	WB				
命令2					F	D	E	WB

$$\text{平均命令実行時間 (CPI)} = \frac{\text{総クロック数}}{\text{実行命令数}} = \frac{8 \text{クロック}}{2 \text{命令}} = 4$$

- (3) 先行制御方式 (命令パイプライン制御方式)

時刻 (クロック)	1	2	3	4	5	6	7	8
命令1	F	D	E	WB				
命令2		F	D	E	WB			
命令3			F	D	E	WB		
命令4				F	D	E	WB	
命令5					F	D	E	WB

$$\text{平均命令実行時間 (CPI)} = \frac{\text{総クロック数}}{\text{実行命令数}} = \frac{8 \text{クロック}}{5 \text{命令}} = 1.6$$

- (4) パイプラインの乱れ ([3] p.97~105 参照)

各ステージの処理が1クロックで終了しない場合

機能回路の競合

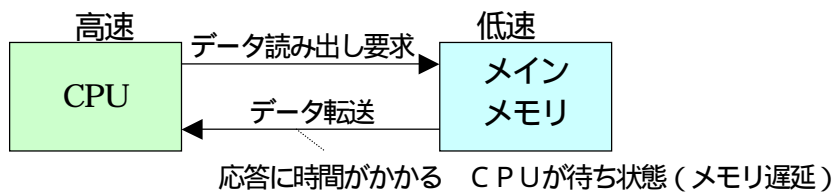
分岐命令による乱れ

命令間のオペランドの依存関係

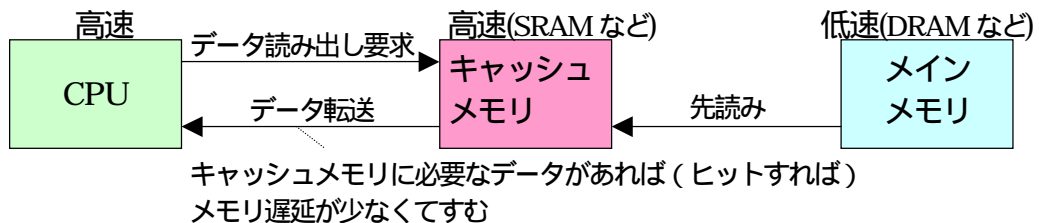
8.4. キャッシュメモリ ([1] p.60~p.66, [3] p.109~p.125)

(1) 仕組み

【キャッシュメモリなし】



【キャッシュメモリあり】



ヒット率 = 必要なデータがキャッシュメモリ内にある確率

【例】 [3] p.109

キャッシュメモリのアクセス時間 = 10 ns

メインメモリのアクセス時間 = 200 ns

ヒット率 = 0.95

平均メモリアクセス時間 = $10 \text{ ns} \times \underline{\quad} + 200 \text{ ns} \times \underline{\quad} = \boxed{19.5 \text{ ns}}$

- ・ キャッシュメモリなし 200 ns に比べて約 10 倍高速
- ・ ヒット率の向上が重要

【ヒット率向上のための方法】 ([1] p.62)

命令キャッシュとデータキャッシュを分離する

(命令とデータの格納場所が異なることが多いため)

複数のキャッシュを用いる

CPU - 1次キャッシュ - 2次キャッシュ - メインメモリ

(2) キャッシュライン (ブロック) とマッピング

キャッシュライン (ブロック) ... キャッシュメモリの単位 (128 ~ 256 bit 程度)

↑ ↓ マッピング (対応づけ、連想方式)

メインメモリ

直接マップ方式 (ダイレクトマッピング (direct mapping) 方式)

対応づけが番地ごとにあらかじめ決定されている方式

完全連想方式 (フルアソシエイティブ (fully associative mapping) 方式)

メインメモリの任意のブロックをキャッシュの任意のブロックにマッピングできる方式

群連想方式 (set associative mapping) と の中間にあたる方式

【例】 ([3] p.113~p.114)

1 番地あたりの記憶容量 1 byte

キャッシュライン (ブロック) の大きさ 1 ブロック = 64 byte = 2^6 byte = 2^6 番地

キャッシュメモリ容量 64 k byte = 2^{16} byte = 2^{10} ブロック = 0 ~ 1023 ブロック

メインメモリ容量 4 G byte = 2^{32} byte = 2^{26} ブロック

直接マップ方式 (direct mapping) の例 ([3] p.114)

メインメモリのブロックとキャッシュメモリのブロックの
対応づけがあらかじめ決定されている方式

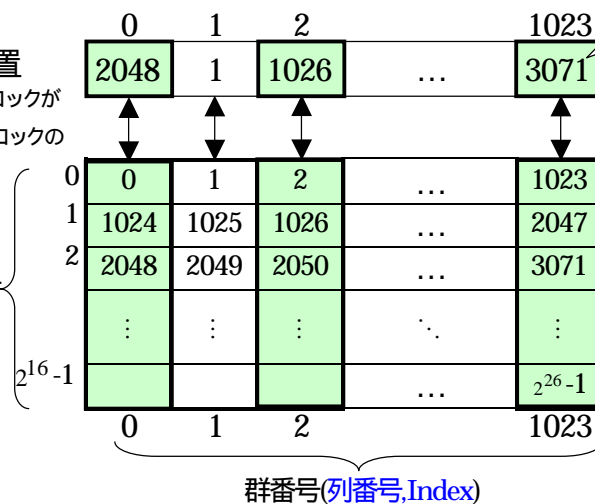
[キャッシュメモリ]

1 行 × 1024 列に配置

キャッシュメモリの1つのブロックが
あらかじめ決められた 2^{16} ブロックの
メインメモリに対応

[メインメモリ]

群内ブロック番号
(行番号, Tag)



ブロック番号 3071
のデータを保持

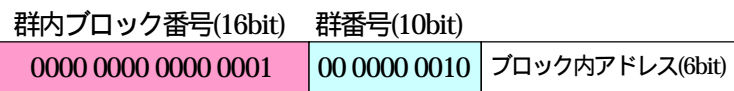
1 ブロック
= 2^6 番地

群番号(列番号, Index)

$$\text{群番号(列番号, Index)} = \text{キャッシュメモリの列数} = 2^{10}$$

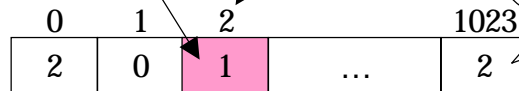
$$\text{群内ブロック番号(行番号, Tag)} = \text{メインメモリ容量} (2^{26} \text{ ブロック}) / \text{群番号} (2^{10}) = 2^{16}$$

【例】メインメモリのアドレス (ブロック番号 1026 のデータが要求された時)



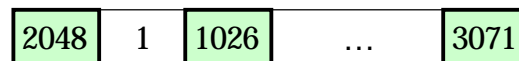
キャッシュメモリにどの群内ブ
ロック番号(行番号)に対応するメイ
ンメモリの内容が記憶されている
かをディレクトリ(連想メモリ)と
呼ばれるレジスタに保存しておく。

[ディレクトリ (連想メモリ)]



群内ブロック番号
(行番号, 16bit)

[キャッシュメモリ]



- 《特徴》
- ・マッピングが簡単
 - ・ヒット率が低下する可能性が高い

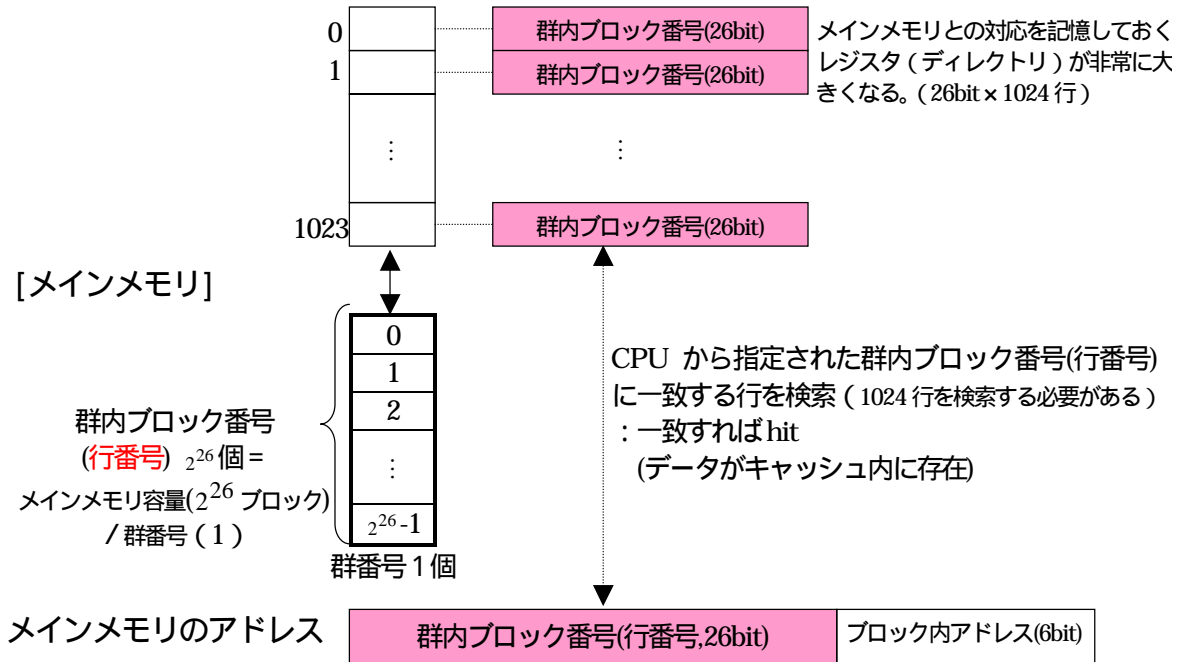
完全連想方式 (fully associative mapping) の例 ([3] p.113)

メインメモリの任意のブロックをキャッシュの任意のブロックにマッピングできる方式

[キャッシュメモリ]

1024 行×1 列に配置

[ディレクトリ (連想メモリ)]



群連想方式 (set associative mapping) の例

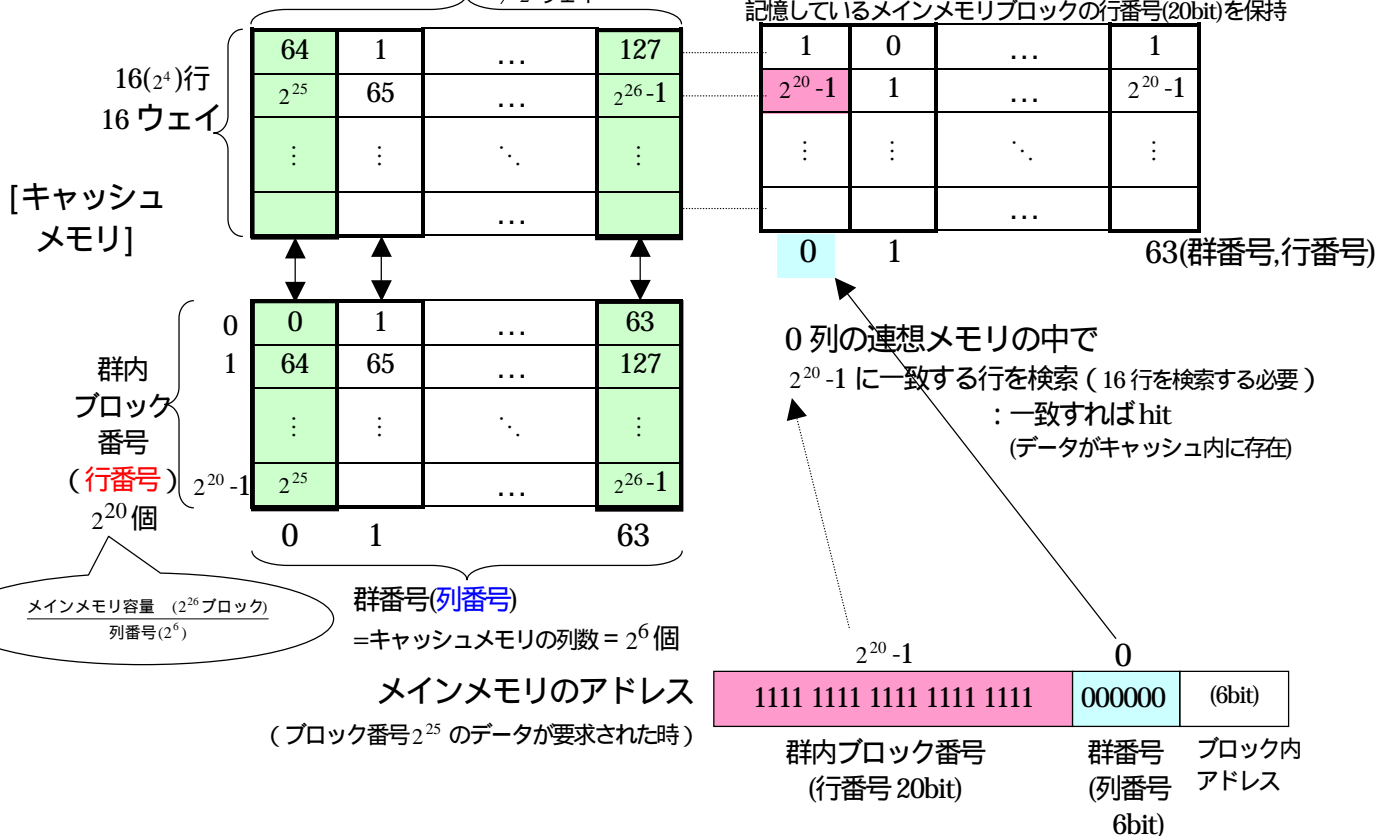
と の中間にあたる方式。キャッシュメモリを n 行 m 列とした場合、各行について完全連想方式、各列について直接マップ方式とする方式。n 行の場合、n ウェイ群連想方式という。

[キャッシュメモリ] 16 行×64 列に配置

(16 ウェイ群連想方式) 2^6 列 = 2^{10} ブロック / 2^4 ウェイ

[ディレクトリ (連想メモリ)]

キャッシュメモリの各ブロックに対応記憶しているメインメモリブロックの行番号(20bit)を保持



【練習】キャッシュメモリの容量が128K バイト、主記憶の最大容量が4G バイトのメモリ系において、ブロックの大きさが32 バイトの8ウェイ set associative mapping を採用したとき、メインメモリブロックの列(群番号、Index)及び行(群内ブロック番号、Tag)の指定に何ビット必要か。またディレクトリ全体で何ビット必要か。([3]p.116 練習問題3.11)

(3) キャッシュメモリの書き込み方式 ([1] p.65)

- ・ **ライトスルー方式**...CPUがキャッシュライン(ブロック)の内容を書き換えたら、メインメモリ上の内容もすぐに書き換える方式
 - 《利点》メインメモリの内容とキャッシュラインとの整合を常にとるため安全
 - 《欠点》書き込み時のメモリ遅延はほとんど改善されない
- ・ **ライトバック方式**...CPUがキャッシュライン(ブロック)の内容を書き換えても、すぐにはメインメモリに対して書き換え要求を出さず、CPUの処理がひまな時を見計らって書き換える方式
 - 《欠点》メインメモリの内容とキャッシュラインとの内容が異なることが多いため、高度なキャッシュ管理システムが必要
 - 《利点》書き込み時のメモリ遅延が発生しにくい

(4) 置き換えアルゴリズム ([3] p.119)

キャッシュメモリ中の空きブロック枠がなくなった場合に、どのブロック枠を置き換えるかを決める方法

- ・ **乱数法**...順番にブロック枠を選ぶ
 - n ウェイの場合、n回の書き換えの間、キャッシュに残るため、ウェイ数が大きければ悪い方法ではない。
 - (一番古いキャッシュを書き換えることになる。first-in first-out)
- ・ **usage bit 法**...各ブロック枠に対応して1bitのusage bitを設け、アクセスがあると1にする。書き換えが必要な場合は、usage bitが0のブロック枠を乱数法で選ぶ。なお、すべてのusage bitが1になったら、全てのusage bitを0にする。

8.5. CISC と R I S C ([1] p.66~p.68)

- (1) **CISC** (Complex Instruction Set Computer, 複合命令セットコンピュータ)
 - 1個の命令で複雑な機能を実現する命令セットを備えたCPU
- (2) **RISC** (Reduced Instruction Set Computer, 縮小命令セットコンピュータ)
 - 使用頻度の高い単純な命令に限定したCPU
- (3) RISC で処理を行う CISC
 - 命令はCISCでありながら、内部的にはRISCで処理を行う
 - CISC命令 内部でRISC命令に変換 RISCコアで命令実行

	CISC	RISC	RISC+CISC
利点	高度な機能を数少ない命令で実現可能	高速 ・1命令当たりの実行時間が大幅に短縮 ・パイプライン効率も良好 ・ハードウェアのみでデコード可能	高速 CISC 命令との互換性あり
欠点	高速化が難しい ・1命令当たりのクロック数が増加 ・回路が複雑化	限定された命令しか使用できない CISC 命令との互換性がない	
制御方式	マイクロプログラム	布線論理 (ワイヤドロジック)	マイクロプログラム +ワイヤドロジック

8.6. 制御アーキテクチャ

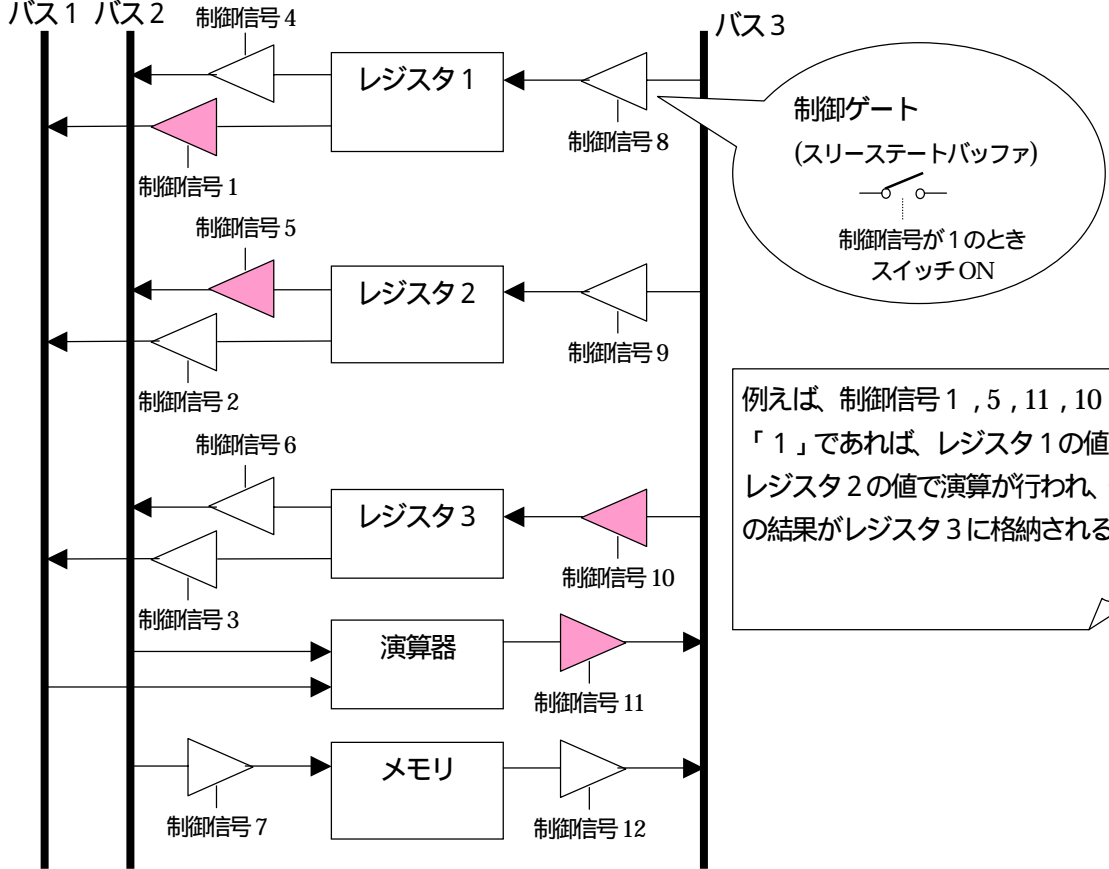
(1) 制御回路の役割

PC メモリ(命令データ) IR (命令レジスタ)

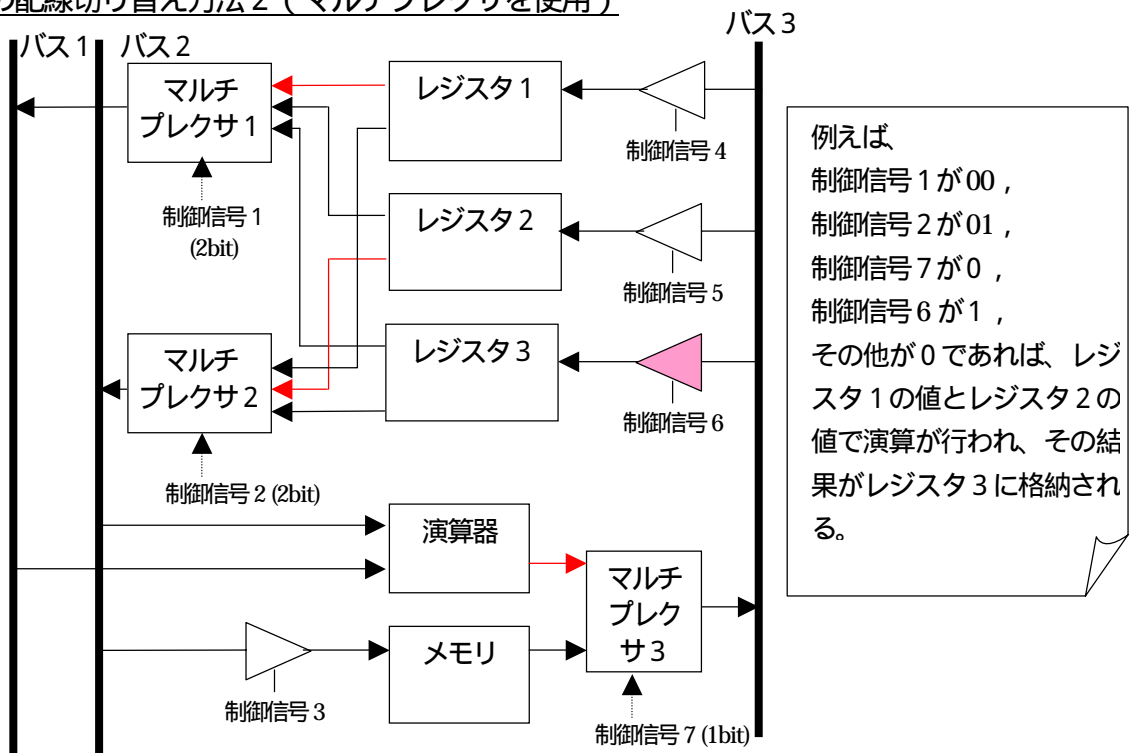
制御回路 (命令のデコード: 命令解読)

制御信号により各部の配線を切り替え

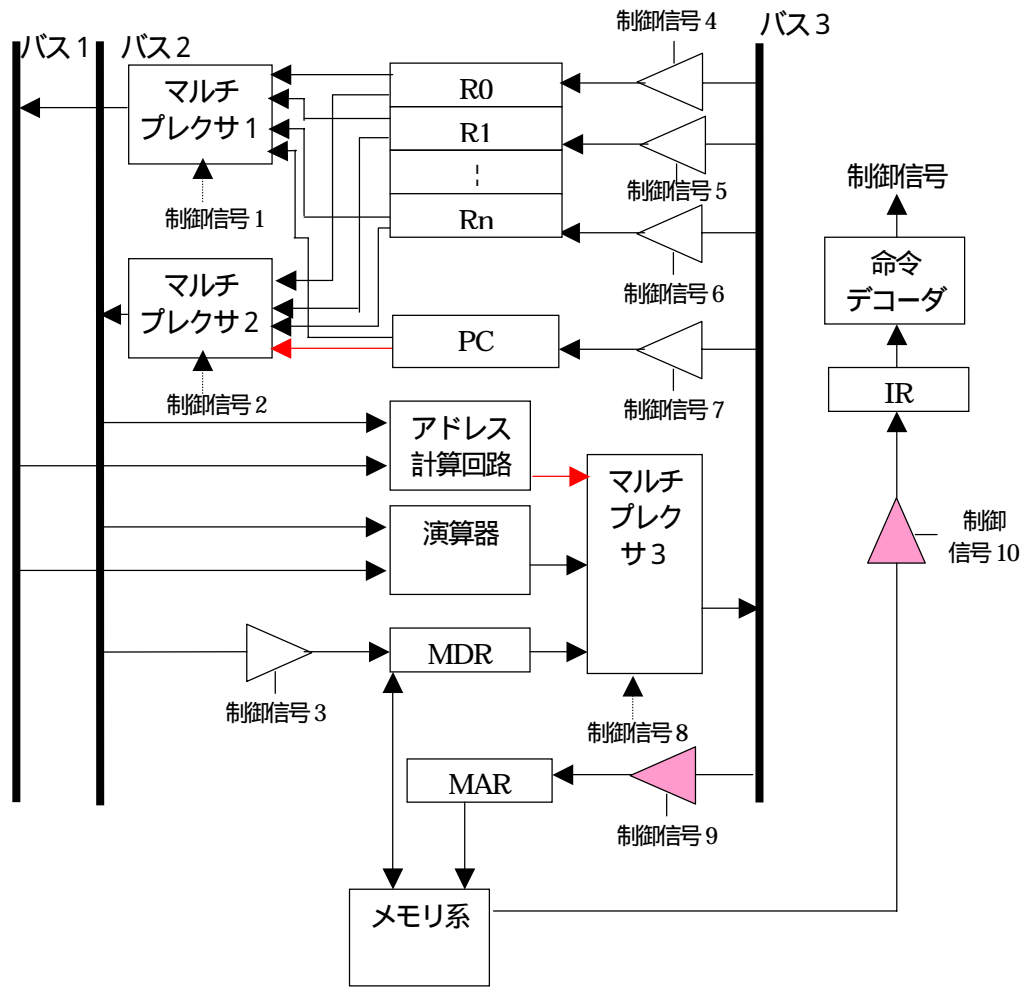
各部の配線切り替え方法1 (制御ゲートを使用)



各部の配線切り替え方法2 (マルチプレクサを使用)



一般的な構成例



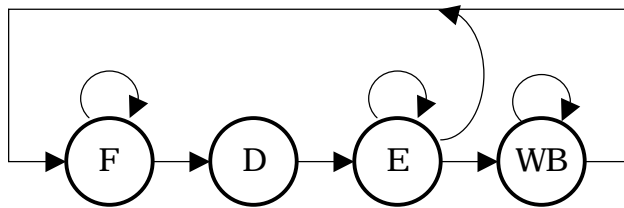
(2) 状態遷移図

- 命令サイクル例 F : 命令のフェッチ (命令の取り出し)
 D : 命令のデコード (命令の解読)
 E : 有効アドレスの計算と命令の実行
 WB : 演算結果をメインメモリに書き戻す

制御信号の系列

- F 制御信号2が1 1、制御信号8が0 0、制御信号9が1、制御信号10が1
 PC マルチプレクサ2 バス2 アドレス計算回路 マルチプレクサ3 MAR メモリ IR
 D IR 命令デコーダ 制御信号 (命令に従い制御信号の時系列が生成)
 E 実行

命令によって変化
 CPUの内部状態によって変化
 メモリ系の状態によって変化



状態遷移図

(3) 制御方式

・布線論理(ワイヤドロジック)制御方式

制御用論理回路を直接ハードウェアによる順序回路で実現する方式

【利点】高速

【欠点】制御回路の設計や修正が困難

RISC で用いられることが多い

・マイクロプログラム制御方式

制御用論理回路をマイクロプログラム (firmware; ファームウェア) という一種のプログラム (ソフトウェア) によって制御する方式

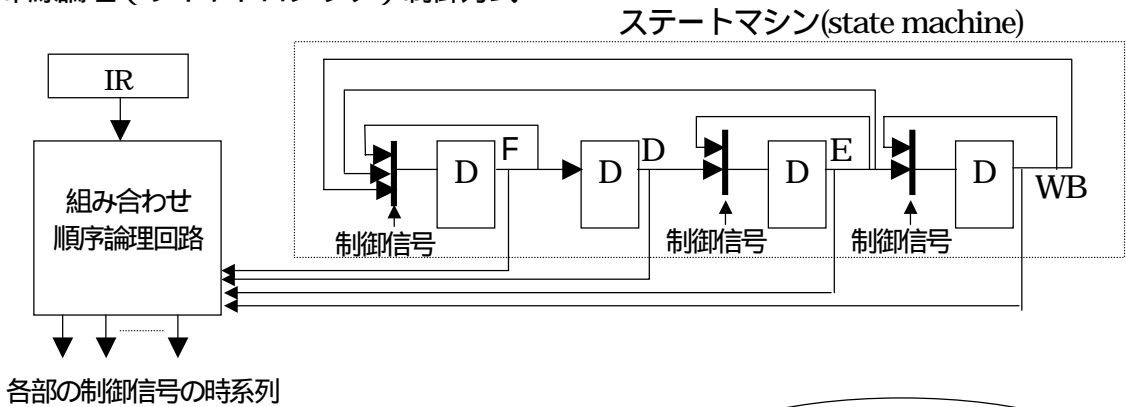
【利点】制御回路の設計や修正が容易

ハードウェアとソフトウェアの役割分担の調整が容易

【欠点】布線論理制御方式より低速

CISC で用いられることが多い

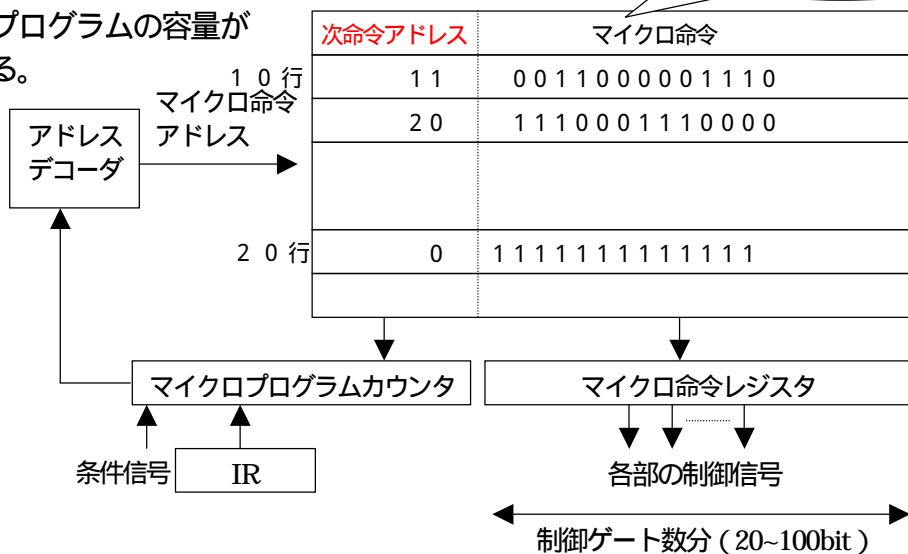
(4) 布線論理(ワイヤドロジック)制御方式



(5) マイクロプログラム制御方式

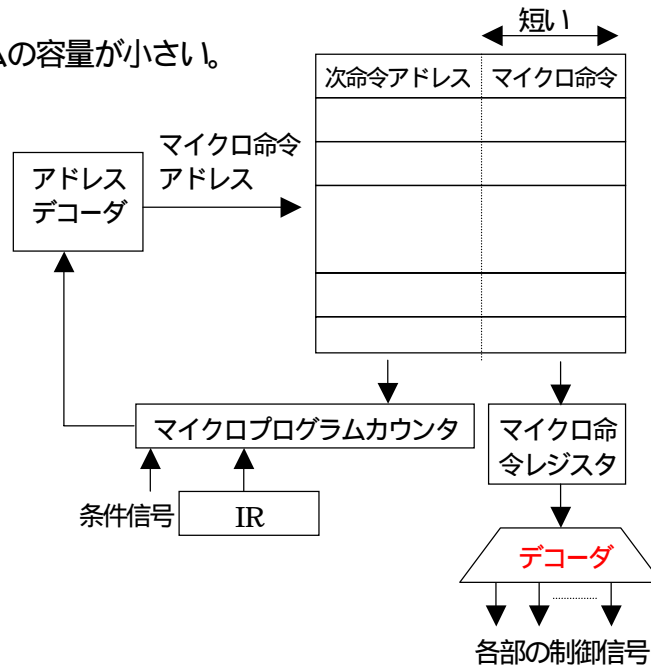
【水平型】

- ・マイクロプログラムの容量が大きくなる。



【垂直型】

- ・ マイクロプログラムの容量が小さい。
- ・ 水平型より低速。



(6) 仮想マシン

マイクロプログラム制御コンピュータの制御メモリが書き換え可能であれば、マイクロプログラムを書き換えることによって、命令セット等を変更することができる。書き換えたマイクロプログラムによって**様々なマシンをエミュレーション (模擬)**することができる。このようなコンピュータを仮想マシンという。

(7) 制御回路

同期式： 無駄時間あり

回路が簡単で、設計保守が容易

非同期式：高速

ハザードが問題となるため、回路が複雑になり、設計保守が困難

(8) 制御回路の設計手順

制御回路への入出力信号を考え、ブロック図を作成

制御状態(state)を決定

状態遷移図作成

真理値表作成し、論理回路の最適化 (この部分は、自動化プログラムを使用することが多い)

参考文献

- [1] 伊勢雅英「よくわかる最新PCアーキテクチャの基本と仕組み」(秀和システム)
- [2] 原田益水「マイクロコンピュータのすべて」(電波新聞社)
- [3] 橋本昭洋「計算機アーキテクチャ」(昭晃堂)
- [4] 松島 守「Z- 8 0 アセンブラ・シミュレータ SEASON/Win」
<http://www.ishikawa-nct.ac.jp/lab/E/www/matsu.html>

付録 命令コード表

(1) データ転送命令 1

r \	A	B	C	D	E	H	L	(HL)	メモ
LD A, r	7F	78	79	7A	7B	7C	7D	7E	$A \leftarrow r$
LD B, r	47	40	41	42	43	44	45	46	$B \leftarrow r$
LD C, r	4F	48	49	4A	4B	4C	4D	4E	$C \leftarrow r$
LD D, r	57	50	51	52	53	54	55	56	$D \leftarrow r$
LD E, r	5F	58	59	5A	5B	5C	5D	5E	$E \leftarrow r$
LD H, r	67	60	61	62	63	64	65	66	$H \leftarrow r$
LD L, r	6F	68	69	6A	6B	6C	6D	6E	$L \leftarrow r$
LD (HL), r	77	70	71	72	73	74	75		$HL \leftarrow r$
LD r, B ₂	3E	06	0E	16	1E	26	2E	36	$r \leftarrow 1$ バイトの数値 B ₂

(2) 演算命令

r \	A	B	C	D	E	H	L	(HL)	メモ
ADD A, r	87	80	81	82	83	84	85	86	$A \leftarrow A + r$
ADC A, r	8F	88	89	8A	8B	8C	8D	8E	$A \leftarrow A + r + CY$
SUB r	97	90	91	92	93	94	95	96	$A \leftarrow A - r$
SBC A, r	9F	98	99	9A	9B	9C	9D	9E	$A \leftarrow A - r - CY$
AND r	A7	A0	A1	A2	A3	A4	A5	A6	$A \leftarrow A \text{ AND } r$
XOR r	AF	A8	A9	AA	AB	AC	AD	AE	$A \leftarrow A \text{ Ex-OR } r$
OR r	B7	B0	B1	B2	B3	B4	B5	B6	$A \leftarrow A \text{ OR } r$
CP r	BF	B8	B9	BA	BB	BC	BD	BE	A - r フラグのみ変化
INC r	3C	04	0C	14	1C	24	2C	34	$r \leftarrow r + 1$
DEC r	3D	05	0D	15	1D	25	2D	35	$r \leftarrow r - 1$

(3) 数値演算命令

Z80	コード	メモ
ADD A, B ₂	C6	$A \leftarrow A + B_2$
ADC A, B ₂	CE	$A \leftarrow A + B_2 + CY$
SUB B ₂	D6	$A \leftarrow A - B_2$
SBC A, B ₂	DE	$A \leftarrow A - B_2 - CY$
AND B ₂	E6	$A \leftarrow A \text{ AND } B_2$
XOR B ₂	EE	$A \leftarrow A \text{ Ex-OR } B_2$
OR B ₂	F6	$A \leftarrow A \text{ OR } B_2$
CP B ₂	FE	A - B ₂ フラグのみ変化

(5) データ転送命令 2

Z80	コード	メモ
LD (BC), A	02	$(BC) \leftarrow A$
LD A, (BC)	0A	$A \leftarrow (BC)$
LD (DE), A	12	$(DE) \leftarrow A$
LD A, (DE)	1A	$A \leftarrow (DE)$
LD (B ₃ B ₂), A	32	$(B_3B_2) \leftarrow A$
LD A, (B ₃ B ₂)	3A	$A \leftarrow (B_3B_2)$

(4) 2バイトデータの命令

r ₁ r ₂ \	BC	DE	HL	SP	メモ
LD r ₁ r ₂ , B ₃ B ₂	01	11	21	31	$r_1r_2 \leftarrow B_3B_2$
ADD HL, r ₁ r ₂	09	19	29	39	$HL \leftarrow HL + r_1r_2$
INC r ₁ r ₂	03	13	23	33	$r_1r_2 \leftarrow r_1r_2 + 1$
DEC r ₁ r ₂	0B	1B	2B	3B	$r_1r_2 \leftarrow r_1r_2 - 1$

(6) データ転送命令 3

Z80	コード	メモ
LD (B ₃ B ₂), HL	22	$(B_3B_2) \leftarrow L$ $(B_3B_2+1) \leftarrow H$
LD HL, (B ₃ B ₂)	2A	$L \leftarrow (B_3B_2)$ $H \leftarrow (B_3B_2+1)$
EX (SP), HL	E3	$(SP) \leftrightarrow L$ $(SP+1) \leftrightarrow H$
EX DE, HL	EB	$DE \leftrightarrow HL$
JP (HL)	E9	$PC \leftarrow HL$
LD SP, HL	F9	$SP \leftarrow HL$

(7) ジャンプ命令

Z80	コード
JP B ₃ B ₂	C3
JP Z, B ₃ B ₂	CA
JP NZ, B ₃ B ₂	C2
JP C, B ₃ B ₂	DA
JP NC, B ₃ B ₂	D2
JP PE, B ₃ B ₂	EA
JP PO, B ₃ B ₂	E2
JP M, B ₃ B ₂	FA
JP P, B ₃ B ₂	F2

(8) コール命令

Z80	コード
CALL B ₃ B ₂	CD
CALL Z, B ₃ B ₂	CC
CALL NZ, B ₃ B ₂	C4
CALL C, B ₃ B ₂	DC
CALL NC, B ₃ B ₂	D4
CALL PE, B ₃ B ₂	EC
CALL PO, B ₃ B ₂	E4
CALL M, B ₃ B ₂	FC
CALL P, B ₃ B ₂	F4

(9) リターン命令

Z80	コード
RET	C9
RET Z	C8
RET NZ	C0
RET C	D8
RET NC	D0
RET PE	E8
RET PO	E0
RET M	F8
RET P	F0

フラグレジスタ

S	Z	0	AC	0	P	1	CY
MSB				LSB			

(10) プッシュ・ポップ命令

	r ₁ r ₂	BC	DE	HL	AF
PUSH	r ₁ r ₂ ,	C5	D5	E5	F5
POP	r ₁ r ₂	C1	D1	E1	F1

(11) 回転命令

Z80	コード	メモ
RLCA	07	CY ← A7 A0 ← A7
RRCA	0F	CY ← A0 A7 ← A0
RLA	17	CY ← A7 A0 ← CY
RRA	1F	CY ← A0 A7 ← CY

(12) 入出力・割り込み命令

Z80	コード	メモ
OUT (B ₂), A	D3	(IO B ₂) ← A
IN A, (B ₂)	DB	A ← (IO B ₂)
DI	F3	割り込み禁止
EI	FB	割り込み許可
RIM	20	A ← マスクの状態
SIM	30	マスクの状態 ← A

(13) その他の命令

Z80	コード	メモ
DAA	27	BCD コード ← A
CPL	2F	A ← A の反転
SFC	37	CY ← 1
CCF	3F	CY ← CY の反転
NOP	00	No Operation
HALT	76	STOP

(14) アセンブラの擬似命令

擬似命令	メモ
ORG	プログラム・データの先頭番地を示す。
END	プログラムの最後につける。
EQU	ラベルに数値を割り当てる。
DB	1バイト単位のデータを定義する。
DW	2バイト単位のデータを定義する。
DS	メモリ領域をバイト単位で確保する。