

# 機械語入門

## コンピュータ用語

### 文字コード

### n進数

コンピュータプログラムをしていると1つの数字を2進数、10進数、16進数と様々な表現で表すことがある。ここで確認しておきましょう。

### 10進数

0から9までの数字をつかい1つの桁を表します。通常一般的によく使用する数値表現です。

コンピュータの数値の最小単位をビットといいます。0と1の2つで表します。

### 2進数

コンピュータの数値の最小単位をビットといいます。0と1の2つで表します。

### 16進数

### ニブル

あまり最近では聞かないのですがニブルという用語があり4ビット単位に区切ります。

8の桁	4の桁	2の桁	1の桁
0	0	0	0

1 + 2 + 4 + 8 = 15 (0xf)

8の桁	4の桁	2の桁	1の桁
1	1	0	0

4 + 8 = 12 (0xc)

### 0x00 のビットイメージ

bit7	bit6	bit5	bit4		bit3	bit2	bit1	bit0
0	0	0	0		0	0	0	0

ビットをニブル単位(4つに)区切って考えると計算しやすくなります。

上位の4bitは0 下位の4bitは0なので 0x00となります。

### 0xa5 のビットイメージ

bit7	bit6	bit5	bit4		bit3	bit2	bit1	bit0

1	0	1	0		0	1	0	1
---	---	---	---	--	---	---	---	---

上位の4bitは10で16進数では0xa、下位の4bitは5なので 0xa5となります。

### 0xff のビットイメージ

bit7	bit6	bit5	bit4		bit3	bit2	bit1	bit0
1	1	1	1		1	1	1	1

上位の4bitは  $1 + 2 + 4 + 8 = 15$  (0xf)  
 下位の4bitも  $1 + 2 + 4 + 8 = 15$  (0xf) なので  
 255 で 16進数では0xFFとなります。

### 1 Bytesでbit7が符号ありとなしの場合

16進数	10進数(符号なし)	10進数(符号あり)
0x0	0	0
0x1	1	1
0x2	2	2
0x3	3	3
0x7F	127	127
0x80	128	-128
0x81	129	-127
0x82	130	-126
0x83	131	-125
0xFC	252	-4
0xFD	253	-3
0xFE	254	-2
0xFF	255	-1

## ハードウェア

### CPU

#### 80386のレジスタ

レジスタ	意味
EAX	アキュムレータレジスタ
EBX	ベースレジスタ



アドレス	+15	+14	+13	+12	+11	+10	+9	+8	+7	+6	+5	+4	+3
0x0000000	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000010	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000020	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000030	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000040	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000050	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000060	00	00	00	00	00	00	00	00	00	00	00	00	00
0x0000070	00	00	00	00	00	00	00	00	00	00	00	00	00
<											>		

## 命令表

コード	名前	意味
00	add	
03 FF	add edi,edi	
08	or	
10	adr	
18	sbb	
20	and	
28	sub	
30	xor	
33 C0	xor eax,eax	eaxレジスタに0を代入する
33 C1		
33 D2	xor edx,edx	edxレジスタに0を代入する
3B C2	cmp eax,edx	eaxとedxを比較する
3B EC	cmp ebp,esp	ebpとespを比較する
40	inc	レジスタの内容を+1する
42	inc edx	edxレジスタの内容を+1する
48	dec	レジスタの内容を-1する
4D	dec ebp	ebpレジスタの内容を-1する
50	push eax	eaxレジスタをスタックにpushする
53	push ebx	ebxレジスタをスタックにpushする

54	push esp	
55	push ebp	
56	push esi	
57	push edi	
5B	pop ebx	
5C		
5D	pop ebp	
5E	pop esi	
5F	pop edi	
68	push \$????????	
7E ??	jle \$??	
7F ??	jg \$??	
81 C4	add esp,\$??	
83 C4	add esp,??	
85 C0	test eax,eax	
8B 01	mov eax,dw ord ptr [ecx]	
8B 40 ??	mov eax,dw ord ptr [eax+\$??]	
8B C8	mov ecx,eax	
8B CE	mov ecx,esi	
8B E5	mov esp,ebp	
8B EC	mov ebp,esp	
8B F0	mov esi,eax	
8D BD	lea edi,\$????????	
90	nop	何もしない
A0	mov al	
B8	mov eax,\$????????	
B9	mov ecx,\$????????	
BA	mov edx,\$????????	
BF	mov edi,1\$????????	
C3	ret	
D1	shl dw ord ptr [ebp-8],1	
D1 FF	sar edi,1	

E0	loopnz ecx	ecxを-1して0以外だったらPCを移動する
E8 ?? ?? ?? ??	call \$????????	
E9	jmp \$????????	
FF 10	call dw ord ptr [eax]	

#### 参考URL

[coder32 edition | X86 Opcode and Instruction Reference 1.12](#)

## load / store

### 演算

- 加減算
- 比較
- シフト命令
- 論理演算
- ビット演算

### ジャンプ、分岐命令

- jp, jz

### サブルーチン呼び出し

- call,ret命令

### 待避・復帰命令

- push,pop  
アルゴリズムで出てくるスタック構造です。

### 入出力命令

- IN, OUT

### 割り込み

- NMI, V-Blank

### IOポート

## C/C++言語を逆アセンブルする

---

### 最初のプログラムはmainから起動する？

- main関数の前に動いているプログラムがあることを説明

### Windows コンソールアプリケーションを作る

1. Visual Studio 2019を起動します
2. メニューの[ファイル(F)]-[新規作成(N)]-[プロジェクト(P)...]を選択
3. [Visual C++]-[Windows コンソールアプリケーション]を選択

ファイル名(N): ConsoleApplication1  
場所(L): 任意  
ソリューションのディレクトリを作成(D) のチェックをつける

## 簡単なC言語のプログラムを逆アセンブラしてみよう

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

## ソースコードにブレークポイントを設定

- int main()の横にマウスカーソルを合わせて赤いマークを付ける  
又はソースコードにカーソルを合わせ

[メニュー]-[デバッグ]-[ブレークポイントの設定/解除]で設定します。  
※F9 キーで設定可能です。

## Debugビルドして 逆アセンブルを確認

1. Debug と x86を選択して 開始又はローカルWindows デバッガを起動
2. ソースコード上でマウス右メニューで逆アセンブルへ移動を選ぶ。
3. 表示オプションのコードバイトの表示を✓をつける

以下の様に表示されます。

```
int main()
{
00A61810 55                push     ebp
00A61811 8B EC            mov      ebp,esp
00A61813 81 EC C0 00 00 00 sub      esp,0C0h
00A61819 53              push     ebx
00A6181A 56              push     esi
00A6181B 57              push     edi
00A6181C 8D BD 40 FF FF FF lea      edi,[ebp-0C0h]
00A61822 B9 30 00 00 00    mov      ecx,30h
00A61827 B8 CC CC CC CC    mov      eax,0CCCCCCCCh
00A6182C F3 AB            rep stos dword ptr es:[edi]
00A6182E B9 02 C0 A6 00    mov      ecx,offset _001BD82A_consoleapplication1@cpp (0A6C002h)
00A61833 E8 DA F9 FF FF    call     @__CheckForDebuggerJustMyCode@4 (0A61212h)
    printf("Hello World!\n");
00A61838 68 30 7B A6 00    push     offset string "Hello World!\n" (0A67B30h)
00A6183D E8 09 F8 FF FF    call     _printf (0A6104Bh)
00A61842 83 C4 04          add      esp,4
}
00A61845 33 C0            xor      eax,eax
00A61847 5F              pop      edi
00A61848 5E              pop      esi
}
00A61849 5B              pop      ebx
00A6184A 81 C4 C0 00 00 00 add      esp,0C0h
```

00A61850 3B EC	cmp	ebp,esp
00A61852 E8 C5 F9 FF FF	call	__RTC_CheckEsp (0A6121Ch)
00A61857 8B E5	mov	esp,ebp
00A61859 5D	pop	ebp
00A6185A C3	ret	

## Releaseビルドして 逆アセンブルを確認

コードが最適化されているのがわかると思います。

printf("Hello World!\n");		
00AE1040 68 F8 20 AE 00	push	offset string "Hello World!\n" (0AE20F8h)
00AE1045 E8 C6 FF FF FF	call	printf (0AE1010h)
00AE104A 83 C4 04	add	esp,4
}		
00AE104D 33 C0	xor	eax,edx
00AE104F C3	ret	

## C言語のプログラムを逆アセンブラしてみよう(その2)

```
#include <stdio.h>

int main()
{
    int number = 0x01234567;
    printf("number=%d", number );
}
```

## Debugビルドして 逆アセンブルを確認

#include <stdio.h>		
#include <memory.h>		
int main()		
{		
010E42F0 55	push	ebp
010E42F1 8B EC	mov	ebp,esp
010E42F3 81 EC CC 00 00 00	sub	esp,0CCh
010E42F9 53	push	ebx
010E42FA 56	push	esi
010E42FB 57	push	edi
010E42FC 8D BD 34 FF FF FF	lea	edi,[ebp-0CCh]
010E4302 B9 33 00 00 00	mov	ecx,33h
010E4307 B8 CC CC CC CC	mov	eax,0CCCCCCCCh
010E430C F3 AB	rep stos	dword ptr es:[edi]
010E430E B9 0A C0 0E 01	mov	ecx,offset _001BD82A_consoleapplication1@cpp (010EC00Ah)
010E4313 E8 FA CE FF FF	call	@__CheckForDebuggerJustMyCode@4 (010E1212h)
int number = 0x01234567;		
010E4318 C7 45 F8 67 45 23 01	mov	dword ptr [number],1234567h
printf("number=%d", number);		
010E431F 8B 45 F8	mov	eax,dword ptr [number]
010E4322 50	push	eax
010E4323 68 30 7B 0E 01	push	offset string "number=%d" (010E7B30h)
010E4328 E8 1E CD FF FF	call	_printf (010E1048h)
010E432D 83 C4 08	add	esp,8
}		
010E4330 33 C0	xor	eax,edx
010E4332 5F	pop	edi
010E4333 5E	pop	esi
010E4334 5B	pop	ebx
010E4335 81 C4 CC 00 00 00	add	esp,0CCh
010E433B 3B EC	cmp	ebp,esp



010E433D E8 DA CE FF FF	call	__RTC_CheckEsp (010E121Ch)
010E4342 8B E5	mov	esp,ebp
010E4344 5D	pop	ebp
010E4345 C3	ret	

## Releaseビルドして 逆アセンブルを確認

```
int number = 0x01234567;
printf("number=%d", number);
00F71040 68 FF 00 00 00      push    1234567h
00F71045 68 F8 20 F7 00      push    offset string "number=%d" (0F720F8h)
00F7104A E8 C1 FF FF FF      call    printf (0F71010h)
00F7104F 83 C4 08            add     esp,8
}
00F71052 33 C0              xor     eax,eax
}
00F71054 C3                ret
```