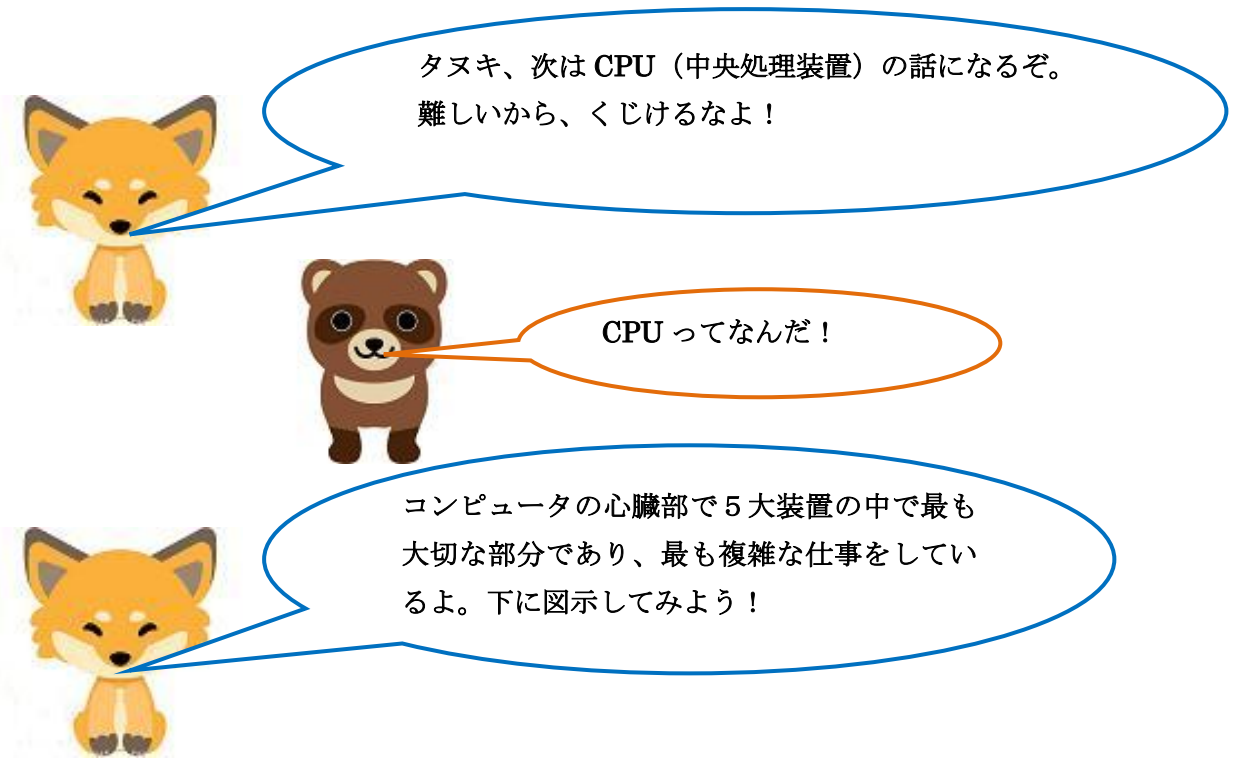
第３話（制御装置と演算装置）



ＣＰＵ（Central　Processing　Unit）

［解　説］

レジスタというのはＣＰＵ内のデータを一時的に記憶する領域に付けられた名称です。

制御装置

・命令アドレスレジスタ

次に実行するプログラムのアドレスを格納するレジスタ。プログラムカウンタ（ＰＲ）ともいう。

・スタックポインタ

主プログラムからサブプログラムに制御を移した時に、戻り先のアドレスを保存するレジスタ。

・フラグレジスタ（３ビット）

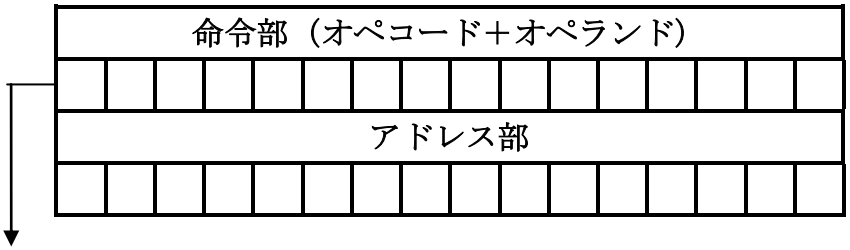
ＯＦ（１ビット）：演算結果が１６ビットの領域に納まらなくなった場合は１、納まる時には０。算術演算、論理演算で異なる。

ＳＦ（１ビット）：演算結果が正（＋）は０、負（－）は１。

ＺＦ（１ビット）：演算結果が０は１、それ以外は０。

・命令レジスタ

以下の形式で、プログラムの命令を格納するレジスタ



・命令解読器（デコーダ）

命令レジスタの命令部のコードを取り出し、解釈し、演算指示を演算装置に伝える。

演算装置

・汎用レジスタ

ＧＲ0からＧＲ4までの５個のレジスタがあります。（数は仕様で異なる）
計算で使用する数値を保存したり、計算結果を保存したりします。
特にＧＲ4は、累算結果を保存するアキュムレータ（累算器）として使われます。

・演算回路

算術（加算・減算・乗算・除算の四則）演算、論理演算、比較演算、シフト演算を行う。

知らない用語が沢山でてきて頭が痛くなってきた。

確かに、タヌキには無理かな。用語は覚えなくても良いから、処理の流れが把握できれば良いんだがな。
ここで出てきた用語を理解する為に、多くの勉強することがあるな、ということに気がついてくれれば良いんだが。
例えば、スタックポインタってなんだ、という疑問を抱いたらスタック領域のＰＵＳＨ，ＰＯＰを勉強しなくてはならないな、という感じだ。これが森を見て木を知る、ということなんだがね！無理かな。

上図の演算回路に加算回路と減算回路があるんだ。前に補数を説明した時に加算回路は簡単で、減算回路は少し複雑で処理が遅い、といったよね。
ここで、それを説明してみようか。

キツネ、図を書いてわかりやすく説明してくれ。

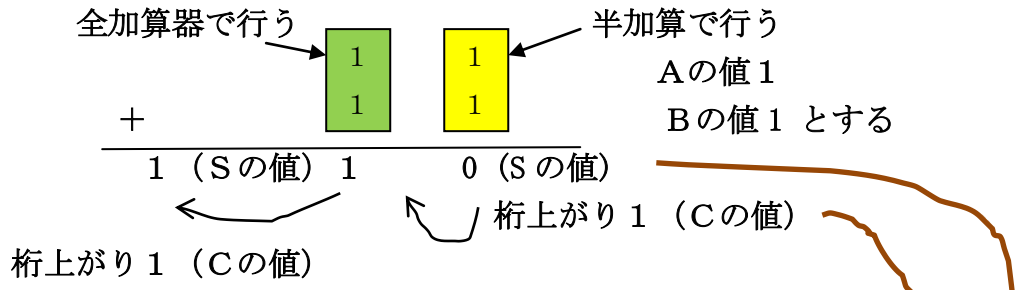
１０進数の３＋３＝６、これならタヌキでもわかるだろう。これを２進数で表すと［１１］＋［１１］＝［１１０］となるんだ。これを加算器で実行させてみよう。加算器には、最初の１桁の加算に使う、半加算器と桁上がりに対応した２桁以降で使う全加算器があることを言っておこう。

演算回路の

加算回路（半加算器と全加算器）

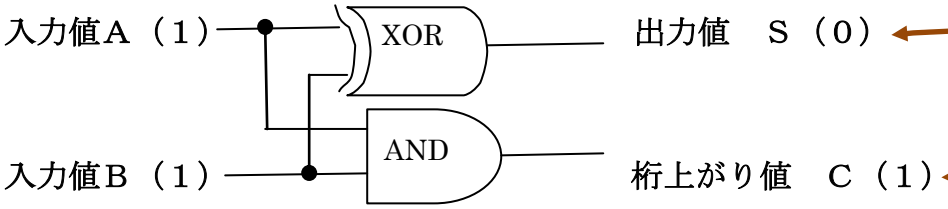
加算器には桁上がりができない半加算器と桁上がりができる全加算器があります。

（計算例）２進数の次の計算をしてみよう



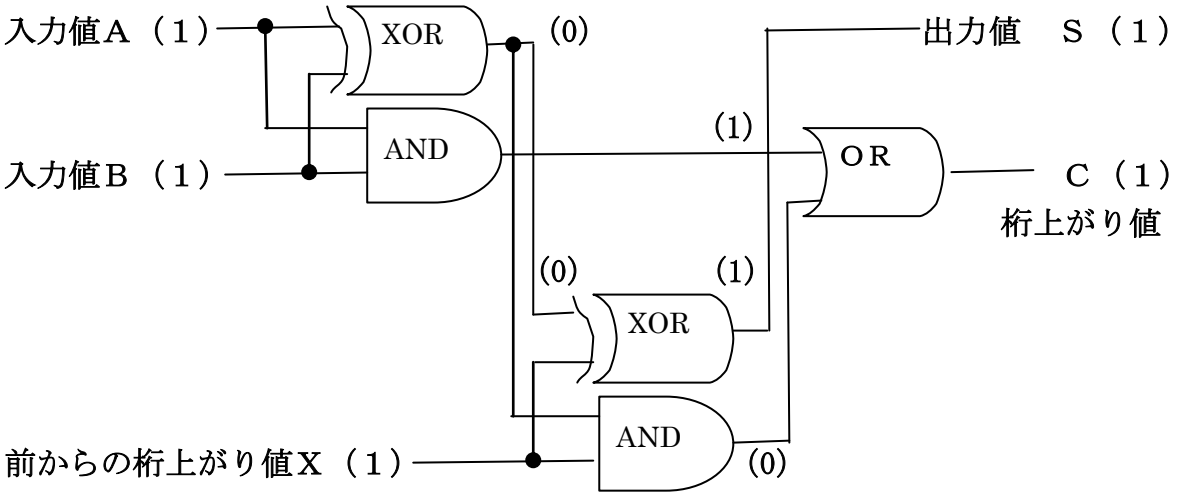
これを半加算器で行わせます。

半加算器（２進数計算の最初の１桁目で利用される）



次の桁（２桁目）を全加算器でさせます。

全加算器（２進数計算の２桁目以降で利用される）



タヌキ、理解できたかな。ここまで来ると、次にＡＮＤ、ＯＲ、ＸＯＲなどの論理演算子を勉強しなければならないな、という事が見えてくるだろ。これが勉強するということなのだ。知れば知るほど、深みにはまっていくんだよ。
さて、減算回路も半減算器と全減算器があり、引算する時には、前から借りてくる、ということが起こり、複雑なので、ここで説明するのは止めるよ！

演算回路については、わかったような、わからないような感じだが、キツネ、他のレジスタとやらの働きが全然見えてこないぞ！

そうなんだ。これじゃわからないよな。
それでオイラ考えた。簡単なプログラムを実行させて、１つ１つ見て行くのが良いのかな、とさ。
ＣＰＵが理解できるプログラム言語は機械（マシン）語という言語なんだ。機械語は最終的には２進数で表現されるんだが、それではあまりにわかりずらいのでアセンブラ言語という言語を使うんだ。アセンブラ言語で記述されたプログラムを機械語に変換することをアセンブルというんだ。

次はプログラム言語の勉強か。

プログラム言語には、Ｃ言語、Python、Java　・・・・
などという言語があるが、最終的には全て機械語に変換されるんだ。ここで、プログラム言語の必要性が出てきただろう。
これから説明に使うのは、アセンブラ言語だからね。アセンブラ言語は、CPU の種類によって仕様が決まる言語なのだ。
それで、基本情報技術者試験で使用されている疑似マシン（COMET）と疑似言語（CASLⅡ）というアセンブラ言語を用いることにするよ。

２０＋（－１０：補数に変換）＝１０という最も簡単な計算をするプログラム（asm1-7.cs2）を考えたんだ。
以下に表示するけれども、言語がわからなくても、感覚的にイメージしてくれれば良いよ。興味を持ったらアセンブラ言語も勉強すれば良いよ。

asml-7.cs2

	アセンブラのソースプログラム		１６進数 機械語（２進数）
	ラベル	命令部（ｵﾍﾟｺｰﾄﾞ+ｵﾍﾟﾗﾝﾄﾞ）	
	REIDAI	START	
	解説	；プログラムの開始	
①		LD GR1,A	1010
		；ｵﾍﾟｺｰﾄﾞ ｵﾍﾟﾗﾝﾄﾞ、ﾗﾍﾞﾙ	8007
		；ﾗﾍﾞﾙAのﾃﾞｰﾀを汎用レジスタ（GR1）に入れる。	
②		ADDA GR1,B	2010
		;GR1の値にﾗﾍﾞﾙBのﾃﾞｰﾀを加算し、結果を GR1 に代入する。	8008
③		ST GR1,ANS	1110
		；ﾗﾍﾞﾙANSに GR1 の値を保存する。	8009
④		RET	8100
		；ﾌﾟﾛｸﾞﾗﾑの停止。	
	A	DC 20	0014
		；ﾃﾞｰﾀは 20	
	B	DC -10	FFF6
		；ﾃﾞｰﾀは-10 だが、補数変換	
	ANS	DS 1	7FFF
		；１語長（16ﾋﾞｯﾄ）の領域確保	
		END	
		；ﾌﾟﾛｸﾞﾗﾑの終了	

主記憶装置

8000-8001 (ｱﾄﾞﾚｽ) ﾌﾟﾛｸﾞﾗﾑ領域

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

8002-8003 (ｱﾄﾞﾚｽ)

0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

8004-8005 (ｱﾄﾞﾚｽ)

0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

8006 (ｱﾄﾞﾚｽ) オペランド無し

1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

8007 (ｱﾄﾞﾚｽ) データ領域

0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0

8008 (ｱﾄﾞﾚｽ) 補数計算されている

1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0

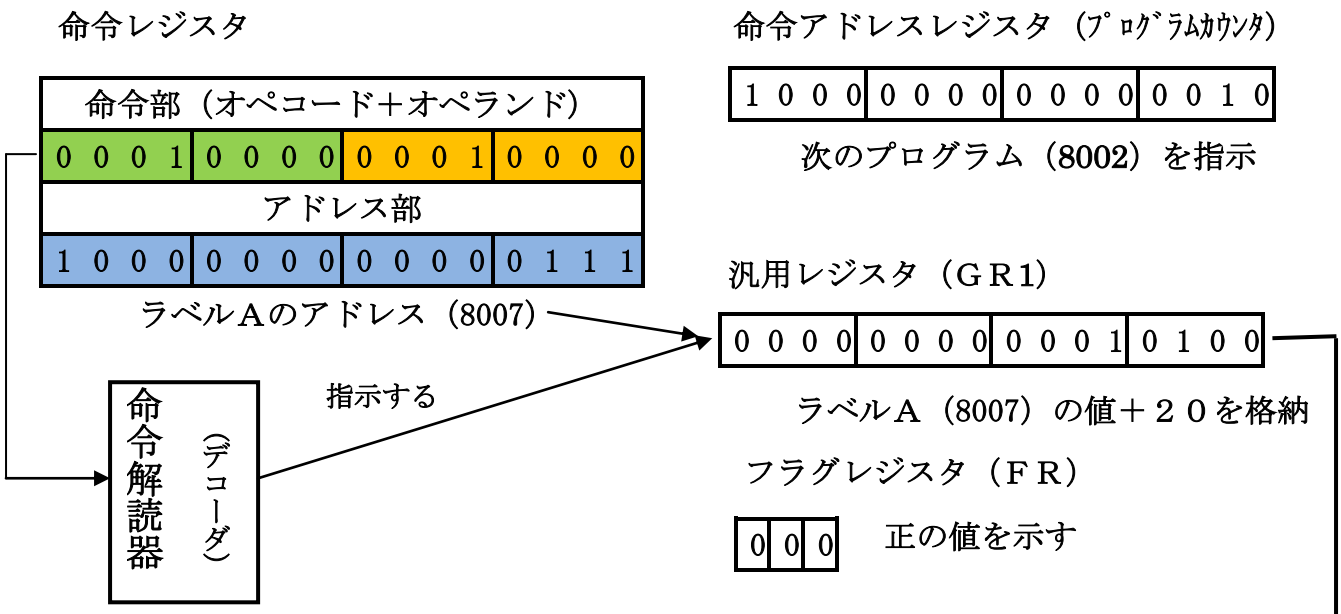
8009 (ｱﾄﾞﾚｽ) １語長確保の初期値

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

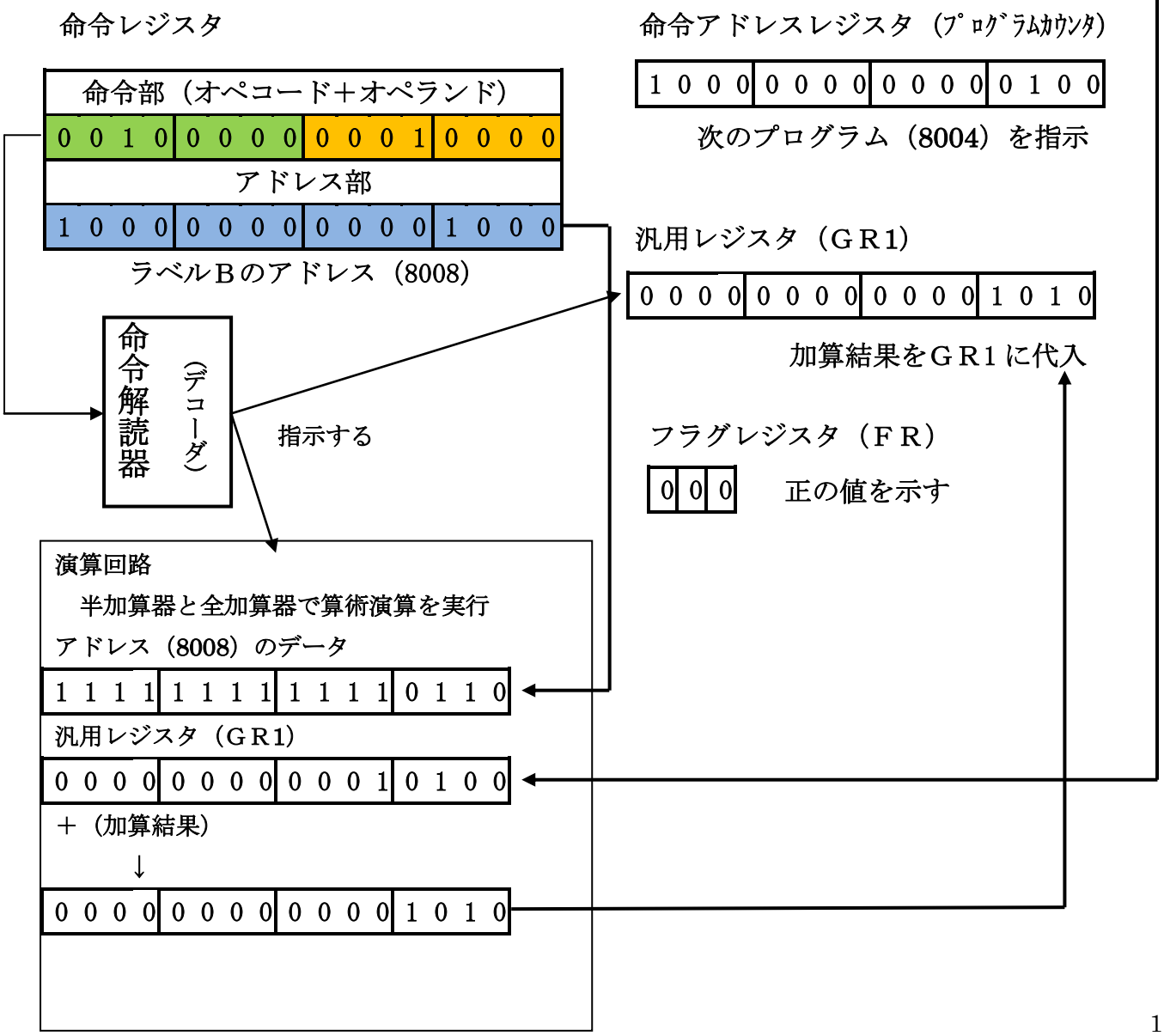
左のソースには、命令部のオペコード（緑色の表示）の解説を付けておいたよ。オペランドの GR1（橙色で表示）は［0001 0000］で表わされるが、GR2 は［0010 0000］となり、使用する汎用レジスタの区別がされています。最初に青で表示している［1000000000000111］（8007）は、ラベルＡの主記憶装置の場所（アドレス）を示しています。赤で表示されてるデータの－１０は２進数の２の補数［1111111111110110］に変換されています。

次は、①から④まで１命令毎に制御装置＆演算装置で処理される状態を見て行くことにしましょう。

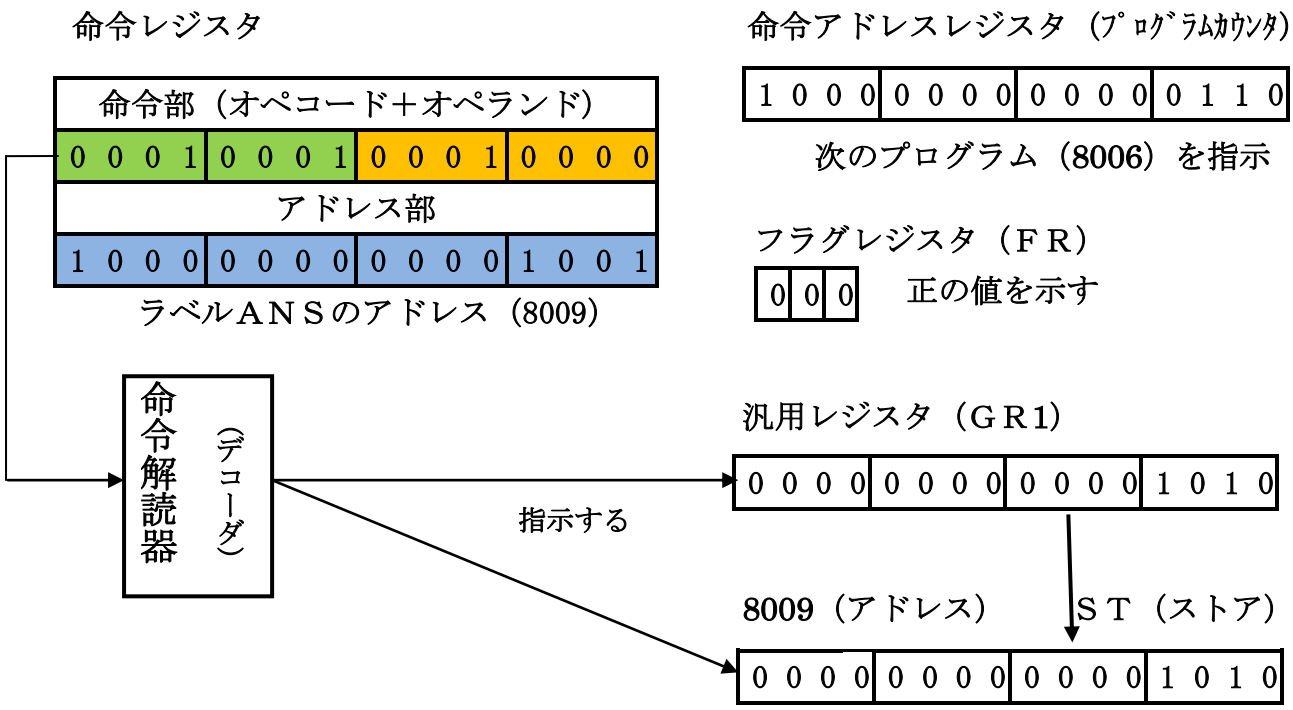
① LD GR1, A



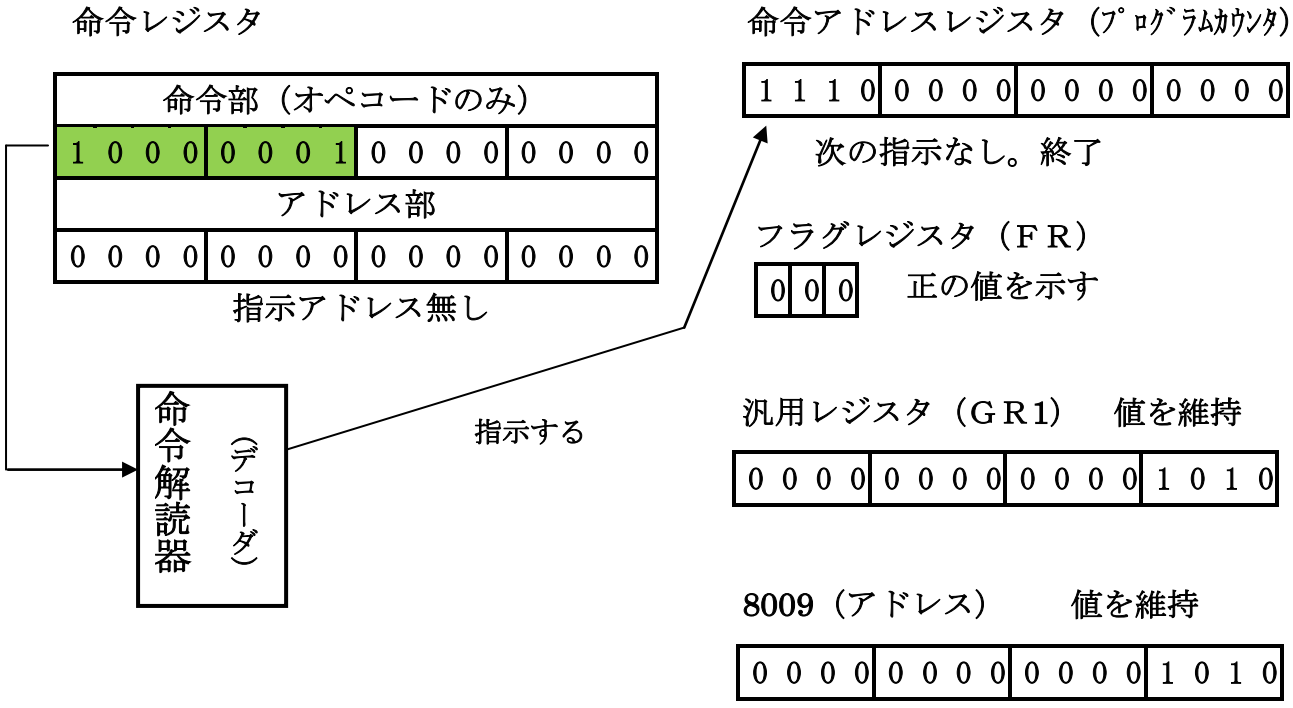
② ADDA GR1, B



③　ＳＴ　ＧＲ1，ＡＮＳ



④　ＲＥＴ



こんな感じだな。タヌキ、理解できた？

わかったような、わからないような。
まあ、何か複雑なことをやっていることはわかった。

アセンブラ言語の勉強もしていないし、
無理ないか。

では、第４話　に進むか。