CASL IIのまとめ (学年末試験にむけて)

山本昌志*

2006年2月24日

1 概要

学年末試験にむけて，後期中間試験からこれまでに学習した内容をまとめる．この間，教科書の p.29〜119 まで学習した．講義の内容は，以下の通りであった．

第8回 アセンブラ命令 (非実行文)
機械語命令の書き方とデータ転送命令

第9回 機械語命令 (算術・論理演算)

第10回 機械語命令 (シフト・比較・分岐)

第11回 機械語命令 (スタック・サブルーチン・他)
マクロ命令

第12回 CASL IIのプログラム例 (その1)
[例題1] 加算
[例題2] 加算と条件分岐
[例題3] マスク処理と条件分岐

第13回 CASL IIのプログラム例 (その2)
[例題4] 論理演算とアドレス修飾
[例題5] シフト演算
[例題6] 繰り返し処理
[例題7] 繰り返し処理とサブルーチン

第14回 CASL IIのプログラム例 (その3)
[例題8] アドレスの受け渡し
[例題9] ラベルを2重につける方法
[例題10] 数値データを文字データに変換

第15回 CASL IIのプログラム例 (その4)
[例題11] 文字データを数値データに変換

*独立行政法人 秋田工業高等専門学校 電気工学科

2 CASL IIの命令語

2.1 アセンブラ命令

教科書の P.28〜P.35 で説明している非実行文と書かれているものである．アセンブラーという変換プログラムに対して，いろいろな指示を行う命令である．COMET II の CPU の動作の指示は行わない．したがって，この命令は機械語に変換されて特定のビットパターン (1 と 0 の組み合わせ) に変換されることはない．

表 1: アセンブラ命令一覧

機能	書式	動作内容	フラグレジスタの変化
プログラム開始	START [実行開始番地]	プログラムの開始を示す．プログラムの最初に，必ず書かなくてはならない．	
プログラム終了	END	プログラムの終わりを示す．ラベルは使えない．プログラムの最後に，必ず書かなくてはならない．	
定数格納	DC n	10 進定数をラベルのアドレスに格納	
	DC #h	16 進定数をラベルのアドレスに格納	
	DC ’文字列’	文字列をラベルのアドレスから格納	
	DC ラベル名	ラベル名が示すアドレスを格納	
領域の確保	DS n	ラベル名で示すアドレスから n 語領域を確保	

注意

- アセンブラ命令ではフラグレジスタの値はセットされることはない．これは，アセンブラ命令はプログラム実行には動作しないためである．

2.2 機械語命令

教科書の P.40〜P.82 で説明している．この命令は，COMET II の CPU の動作の指示を行う．そのため，この命令に対応した論理回路が，CPU の中に組み込まれている．これら命令は，アセンブラーにより特定のビットパターンの機械語に変換され，そのパターンに従い，論理回路が動作する．

表 2: データの転送命令

機能	書式	動作内容	フラグレジスタの変化
ロード	LD r1,r2	レジスタ r2 の値をレジスタ r1 にコピー	コピーされた値に従い以下のようになる． OF 0:常にゼロが設定される SF 1:負の時 (第 15 ビットが 1) 0:正の時 (第 15 ビットが 0) ZF 1:ゼロの時 (全てのビットが 0) 0:ゼロ以外
	LD r,adr[,x]	アドレス adr[,x] の主記憶の内容をレジスタ r にコピー	
ストア	ST r,adr[,x]	レジスタ r の内容を主記憶装置のアドレス adr[,x] にコピーする	変化なし
ロードアドレス	LAD r,adr[,x]	主記憶装置のアドレス値 adr[,x] をレジスタ r にコピーする．	変化なし

表 3: 演算命令

機能	書式	動作内容	フラグレジスタの変化
算術加算	ADDA r1,r2	レジスタ r1 と r2 の符号付き加算 r1←r1+r2	演算結果の値に従い以下のようになる. OF 1:結果が-32768〜32767 の範囲外 0:範囲内 SF 1:負 (第 15 ビットが 1) 0:正 (第 15 ビットが 0) ZF 1:ゼロ (全てのビットが 0) 0:ゼロ以外
	ADDA r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容を符号付加算 r←r+adr[,x] の内容	
算術減算	SUBA r1,r2	レジスタ r1 と r2 の符号付き減算 r1←r1-r2	
	SUBA r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容を符号付減算 r←r-adr[,x] の内容	
論理加算	ADDL r1,r2	レジスタ r1 と r2 の符号無し加算 r1←r1+r2	演算結果の値に従い以下のようになる. OF 1:結果が 0〜65535 の範囲外 0:範囲内 SF 1:第 15 ビットが 1) のとき 0:第 15 ビットが 0) のとき ZF 1:ゼロ (全てのビットが 0) 0:ゼロ以外
	ADDL r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容を符号無し加算 r←r+adr[,x] の内容	
論理減算	SUBL r1,r2	レジスタ r1 と r2 の符号無し減算 r1←r1-r2	
	SUBL r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容を符号無し減算 r←r-adr[,x] の内容	
論理積	AND r1,r2	レジスタ r1 と r2 のビット毎の論理積を計算. 結果は r1 に格納.	演算結果の値に従い以下のようになる. OF 0:常にゼロが設定される. SF 1:第 15 ビットが 1) のとき 0:第 15 ビットが 0) のとき ZF 1:ゼロ (全てのビットが 0) 0:ゼロ以外
	AND r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容のビット毎の論理積を計算. 結果は r1 に格納.	
論理和	OR r1,r2	レジスタ r1 と r2 のビット毎の論理和を計算. 結果は r1 に格納.	
	OR r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容のビット毎の論理和を計算. 結果は r1 に格納.	
排他的論理和	XOR r1,r2	レジスタ r1 と r2 のビット毎の排他的論理和を計算. 結果は r1 に格納.	
	XOR r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) の内容のビット毎の排他的論理和を計算. 結果は r1 に格納.	

表 4: 比較命令

機能	書式	動作内容	フラグレジスタの変化
算術比較	CPA r1,r2	レジスタ r1 と r2 を符号付き整数として比較を行う. 比較の結果は, FR に設定.	2 つの整数の比較 (以下の演算) を行う. r1-r2 r-adr[,x] の内容 OF 0:常にゼロが設定される. SF 1:負 (第 15 ビットが 1) のとき) 0:正 (第 15 ビットが 0) のとき) ZF 1:等しい (全てのビットが 0) 0:等しくない
	CPA r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) を符号付き整数として比較. 比較の結果は, FR に設定.	
論理比較	CPL r1,r2	レジスタ r1 と r2 を符号無し整数として比較. 比較の結果は, FR に設定.	
	CPL r,adr[,x]	レジスタ r と主記憶装置 (アドレス adr[,x]) を符号無し整数として比較. 比較の結果は, FR に設定.	

表 5: シフト命令

機能	書式	動作内容	フラグレジスタの変化
算術左シフト	SLA r,adr[,x]	レジスタ r の内容を符号ビットを除き，adr[,x] の番地分，各ビットを左へシフト．空いたビットには 0 が入る．	OF :最後に送り出されたビットの値 SF 1:負の時 (第 15 ビットが 1) 0:正の時 (第 15 ビットが 0) ZF 1:ゼロの時 (全てのビットが 0) 0:ゼロ以外
算術右シフト	SRA r,adr[,x]	レジスタ r の内容を符号ビットを除き，adr[,x] の番地分，各ビットを右へシフト．空いたビットには符号ビットと同じ値が入る．	
論理左シフト	SLL r,adr[,x]	レジスタ r の内容を，adr[,x] の番地分，各ビットを左へシフト．空いたビットには 0 が入る．	
論理右シフト	SRL r,adr[,x]	レジスタ r の内容を，adr[,x] の番地分，各ビットを左へシフト．空いたビットには 0 が入る．	

表 6: 分岐命令

機能	書式	動作内容	フラグレジスタの変化
正分岐	JPL adr[,x]	フラグレジスタの SF と ZF の両方が 0 の時 (比較の結果，正)，adr[,x] のアドレスへ分岐 (実行が移動) する．	変化無し
負分岐	JMI adr[,x]	フラグレジスタの SF が 1 の時 (比較の結果，負)，adr[,x] のアドレスへ分岐 (実行が移動) する．	
非零分岐	JNZ adr[,x]	フラグレジスタの ZF が 0 の時 (比較の結果，等しくない)，adr[,x] のアドレスへ分岐 (実行が移動) する．	
零分岐	JZE adr[,x]	フラグレジスタの ZF が 1 の時 (比較の結果，等しい)，adr[,x] のアドレスへ分岐 (実行が移動) する．	
オーバーフロー分岐	JOV adr[,x]	フラグレジスタの OF が 1 の時 (オーバーフロー)，adr[,x] のアドレスへ分岐 (実行が移動) する．	
無条件分岐	JUMP adr[,x]	無条件に，adr[,x] のアドレスへ分岐 (実行が移動) する．	

表 7: スタック操作命令

機能	書式	動作内容	フラグレジスタの変化
プッシュ	PUSH adr[,x]	スタック領域に，adr[,x] のアドレスを格納する．	変化無し
ポップ	POP r	スタック領域からデータを取りだし，レジスタ r に格納	

表 8: サブルーチンに関する命令

機能	書式	動作内容	フラグレジスタの変化
コール	CALL adr[,x]	サブルーチンを呼び出す．adr[,x] に実行が移る．	変化無し
リターン	RET	サブルーチンから呼び出し元のルーチンへ実行が移る．	

表 9: その他の命令

機能	書式	動作内容	フラグレジスタの変化
スーパーバイザーコール	SVC adr[,x]	OS の機能を呼び出す．マクロ命令の IN や OUT で使われている．	不定．OS に依存する．
ノーオペレーション	NOP	なにも実行されない命令．	変化しない．

2.3 マクロ命令

教科書の P.83〜P.86 で説明している．マクロ命令とは，特定の機能を果たす，いくつかの機械語命令の集まりに名前を付けたものである．この名前を指定するだけで，これらの命令の集まりが実行できる．これにより，頻繁に使われる定形的な命令群をマクロ命令にすることにより，同じようなプログラムをいちいち書くことを省くことができ，便利である．サブルーチンみたいになっている．

表 10: マクロ命令

機能	書式	動作内容	フラグレジスタの変化
入力命令	IN ラベル 1，ラベル 2	入力領域 (ラベル 1) に入力装置から文字データを入れる．入力文字長は，ラベル 2 に入る．	不定．OS に依存
出力命令	OUT ラベル 1，ラベル 2	出力領域 (ラベル 1) の文字データ，ラベル 2 が示す数だけを出力装置に送る．	不定．OS に依存
レジスターの待避	RPUSH	汎用レジスター内容を，GR1→GR7 の順序でスタック領域に格納．	不定．OS に依存
レジスターの復元	RPOP	スタック領域の内容を，GR7→GR1 の順序で汎用レジスタに格納．	不定．OS に依存

3 CASL II のプログラム例

3.1 [例題 1] 加算

FORTRAN や C 言語等の高級言語では，メモリーの中のデータ同士を加算して直接メモリーに格納することができる．FORTRAN で次のように加算したとおりである．

```
C=A+C
```

これに対して，アセンブラ言語ではレジスターを通して加算の処理が必要である．先ほどと同じことをするためには，

```
LD    GR1,A        ; アドレス A の内容を汎用レジスター GR1 にコピー
ADDA  GR1,B        ;GR1 <-  GR1+ B
ST    GR1,C        ;GR1 の内容をアドレス C にコピー
```

と書かなくてはならない．加算に限らず，あらゆる演算や処理にレジスターが関わってくる．

コンピューターというものは，メモリーにあるデータをレジスターにコピーして，CPU で処理して，メモリーにコピーすることを繰り返しているにすぎない．高級言語を用いたプログラムでは，それが見えない

ように隠しているのである．そうすることによりプログラマーの負担を減らしており，その分コンパイラーが頑張っている．アセンブラー言語の場合，CPU の動作そのものを記述する必要があるため，メモリーの処理とかレジスターの動作をその都度書く必要があり，プログラマーは大変である．大変な分，高級言語よりも高速でメモリーが少なくて動作するプログラムを作ることが可能となる．

3.2 [例題 2] 加算と条件分岐

条件分岐とは，ある条件に依存して実行される文が変わるようなプログラム構造を言う．高級言語の場合，条件分岐は実に簡単に実装できる．例えば，変数 a と b の大きい方から小さい方を減算する場合，

```
if(a<b){
  c=b-a;
}else{
  c=a-b;
}
```

と書けばよい．ここで，if 文の括弧の中の演算を制御式と言う．高級言語は，人間が使っている言葉とほとんど同じで，プログラムが簡単に書ける．

しかし，アセンブラーでは，こんなに簡単ではない．そもそも，if 文がないため，それに変わるテクニックを使わなくてはならない．機械語命令を組み合わせて，高級言語の if 文と同じことをするのである．かなりプログラムは面倒であるが，その分コンピューターのハード ウェアー (特に CPU) は簡単になり，高速の動作が可能になる．

アセンブラー言語で if の様な制御文を実現するためには，次のようにする．

1. 制御式の結果をフラグレジスターに設定する．通常 CPA や CPL 命令が使われるが，フラグレジスターが設定できるもので有れば何でも良い．
2. フラグレジスターの値により，分岐する命令 (JPL，JMI，JNE，JZE，JOV，JUMP) を使い，実行する文を選択する．
3. 実行先は，ラベルで指定する．

教科書の [例題 2] では，次のようにしている．

```
ADDA   GR1,B          ;GR1=GR1+B   結果の状態がフラグレジスターにセット
JOV    L1             ;OF(オーバーフローフラグ) が 1 ならラベル 1 へ
JUMP   L2             ; 無条件でラベル L2 へ
```

3.3 [例題 3] マスク処理と条件分岐

データの特定のビットパターンを選び出すことをマスキングという．このビットパターンを選び出すために，演算を行うわけであるが，その演算のためのデータをマスクと言う．例えば，教科書の List5-3 の場

合，2 行目の AND GR0,MASK がマスキング (マスク処理) であって，ラベル A のデータがマスクである．このマスクを用いたマスキングにより，GR0 特定のビットパターンを選び出している．

教科書の例では偶数か奇数か判断するために，最下位ビットをマスクを用いて検査している．ここで用いるマスクパターンは，最下位ビットを調べれば良いので，0000000000000001 となっている．$(1452)_{10}$ のばあい，それは $(0000010110101100)_2$ なので

```
        0000010110101100
AND     0000000000000001
----------------------------
        0000000000000000
```

としている．この例から分かるように，論理積 (AND) の結果は，ラベル A の最下位ビットに依存していることが分かる．最下位ビットの 1 の有無は，フラグレジスタの ZF を見れば分かる．演算の結果，全てのビットがゼロになれば，ZF=1 となる．

データの調べたいビットは，マスクにより指定している．このように，調べたいビットをしているデータマスクという．要するに，お面 (マスク) で顔の一部を隠すように，興味のないビットを隠しているのである．

先の例でも分かるが，AND を使ったマスク処理の場合，マスクは興味の対象のビットを 1，どうでも良いビットを 0 にする．そうすると，興味のないビットは全てゼロとなり，重要なビットは変更されない．先の場合，ある特定の 1 ビットの状態が分かれば良かったので，マスク処理後，直ぐにフラグレジスタ ZF を見た．もう少し複雑な場合は，これではだめである．特定のビットパターンを調べたい場合について考える．例えば，

＊＊＊＊1010＊＊＊＊1100

のような場合である．ここで，＊は 0 でも 1 でもよく，興味の対象外のビットである．

このようなビットパターンを調べる場合，2 つの手順が必要であろう．

1. まず興味の対象のビットを取り出す必要がある．興味の対象外のビットは，0 または 1 に設定する．
2. 興味の対象のビットがある特定のビットパターンになっているか，否か調べる．

これを，AND と OR を使って調べる．

まずは，AND を使う方法である．調べたいデータは GR0 に格納されているとする．

```
                    ; これ以前は省略
    AND  GR0,A      ; マスク
    CPL  GR0,B      ; ビットパターンの比較
                    ; このあたりも省略
A   DC   #0F0F      ; マスク
B   DC   #0A0C      ; ビットパターンの定義
```

同じ様なことが，ブール代数の双対の原理[1]により，OR を使ってもできる．そのほかにも，いろいろな方法が考えられる．

[1] 0 と 1, そして論理和と論理積を入れ替えても同じことが成り立つ

3.4 [例題 4] 論理演算とアドレス修飾

プログラムの命令領域とデータ領域は，図 1 のようになるだろう．プログラムの書き方によっては，こうならないこともあるが，通常はこのようになる．

この場合，プログラムのデータ領域にアクセスする事を考える．ラベル A や B は簡単で，ラベル名を示せば良い．ラベル名はアドレスを示すからである．問題は，結果を格納する領域である．このアドレスは，3 つ続いて確保されているが，先頭だけ ANS とラベル名がある．残りの 2 つの表し方である．これらのアドレスは，ANS+1 と ANS+2 である．ANS のアドレスにオフセットの値を加算するのである．

プログラムで使うメモリーのアドレスは，ANS+オフセットで，オフセットは，0,1,2 とすれば良い．論理和の結果を ANS+0，論理積の結果を ANS+1，論理和の結果を ANS+2 に格納する．プログラムでは，オフセットの 0,1,2 を GR2 に入れておき，

```
ST      GR1,ANS,GR2
```

と書く．演算の結果 (GR1) の値が，ANS にオフセット値 (GR2) を加えたアドレスに格納される．

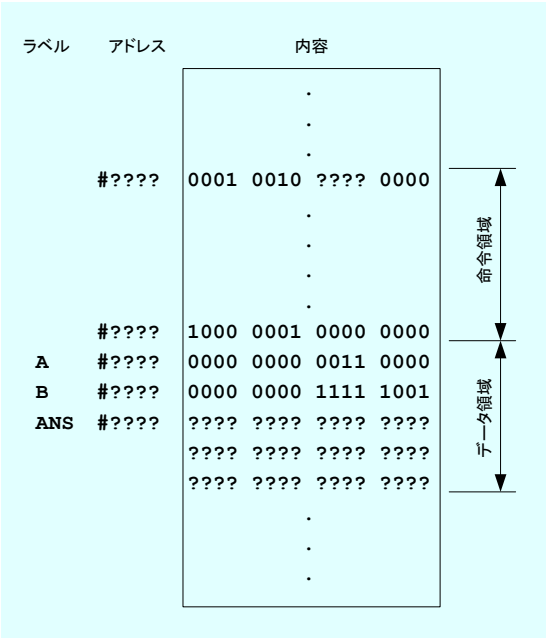

図 1: 教科書の List5-4 のプログラムを実行する場合のメモリ構造．図中の?は値はあるが，不明を示している．

ここで，使っている GR2 のように，1 つずつ値が増加するものをカウンターと呼ぶことがある．これを使うためには，

- カウンターの初期化．ここでは，GR2 をゼロに設定する．
  - CASL では，LAD GR2,0
- カウンターのインクリメント．カウンターの値を 1 増加させる．
  - CASL では，LAD GR2,1,GR2

とする．このテクニックは，重要である．内容をよく理解する必要がある．

3.5 [例題 5] シフト演算

積 (かけざん) の演算を行うとき，シフト命令を使えば効率の良いプログラムができる．シフト命令を使った積の演算は，小学生のときに学習をした筆算の掛け算と同じである．たとえば，34×24 を計算する場合，筆算は $34 \times (2 \times 10^1 + 4 \times 10^0)$ と分解したはずである．そうして，次の手順でこの除算を行ったはずである．

1. 34×2 を計算し，1 桁ずらす (10 倍する)．
2. 34×4 を計算する．

3. 先の計算結果を合計する．この合計 816 が 34×24 の計算結果である．

同じことを 2 進数で行う．これがコンピューターによる乗算である．先ほどと同じ計算 (32×24) を行う．これを 2 進数で表現すると，

$$(100010)_2 \times (11000)_2 = (100010)_2 \times (1 \times 2^4 + 1 \times 2^3)$$

となる．これを先ほど同様の手順で計算する．

1. 掛け算は 1 倍なので計算する必要が無く，最初に $(100010)_2$ を 4 桁左にずらす (ビットシフト)．すると，$(1000100000)_2$ となる．
2. 次に $(100010)_2$ を 3 桁左にずらす．すると，$(100010000)_2$ となる．
3. 先の計算結果を合計すると，$(1100110000)_2$ となる．これは，10 進数の 816 である．

シフトと加算命令でかけ算ができることが分かったはずである．

今回の問題の用に分数の場合でも，

$$\begin{aligned} 0.75 &= \frac{1}{2} + \frac{1}{4} \\ &= \left(2^{-1}\right) + \left(2^{-2}\right) \end{aligned} \tag{1}$$

と分解する．右に 1 ビットシフトさせたものと，右に 2 ビットシフトさせたものを加算すれば良い．これを実現するためには，次のようにプログラムを書けばよい．ラベル A の値を 0.75 倍した結果をラベル KOTAE に格納する．

```
LAD   GR1,0       ; 演算の結果を入れる．初期化
LD    GR2,A       ;A の内容を GR2 へ
SRA   GR2,1       ; 右へ 1 ビットシフト
ADDA  GR1,GR2     ;1 ビットシフトした結果を加算
LD    GR2,A       ;A の内容を GR2 へ
SRA   GR2,2       ; 右へ 2 ビットシフト
ADDA  GR1,GR2     ;2 ビットシフトした結果を加算
ST    GR1,KOTAE   ; 演算結果を KOTAE に
```

教科書のように

$$0.75 = 1 - \left(2^{-2}\right) \tag{2}$$

と分解するのは一般的ではないと思われる．

3.6 [例題 6] 繰り返し処理

高級言語では繰り返し専用の命令が用意されているが，アセンブラ言語にはない．そのため，比較命令 (CPA,CPL) とジャンプ命令 (JMI,JNZ,JZE,JUMP,JPL,JOV) を上手に使って，繰り返し処理を行うことになる．教科書では次のようにしている．

```
LOOP    LAD   GR2,1,GR2   ; 繰り返し処理の始まり
     省略 (いろいろな処理)
SKIP    CPA   GR1,GR2     ; 繰り返し処理終了のためのフラグの設定
        JPL   LOOP        ;GR1-GR2>0 の場合，LOOP へ
```

3.7 [例題 7] 繰り返し処理とサブルーチン

何回も使う処理や複雑な処理はサブルーチンという別のプログラムにするのが，定石である．そうすることにより，プログラムは簡潔に書け，内容が分かりやすくなる．

CASL II の場合，サブルーチンと言う別プログラムは，CALL 命令を使って呼び出す．サブルーチンでの処理が終わると，RET 命令により，呼び出し元へ戻る．教科書の例では，次のようにしている．

```
        CALL  SAIDAI       ; サブルーチン SAIDI の呼び出し
          省略
SAIDAI  LAD   GR1,-1,GR1   ; サブルーチンでの処理の始まり
          省略
        RET                ; サブルーチンでの処理の終わり (呼び出し元へ)
```

サブルーチンは別プログラムではあるが，メモリーやレジスターなどは共有されることに注意が必要である．

3.8 [例題 8] アドレスの受け渡し

利用しやすいサブルーチンを作るためには，その独立性を高めなくてはならない．独立性を高めるためには，データの共有を減らすことである．CASL II の場合，汎用レジスターを使ってデータの受け渡しを行う．また，汎用レジスターはデータの処理にも使われる．サブルーチンでは，できるだけ汎用レジスターの値を変更しないようにする．これを実現するためには，

- サブルーチンで変更した汎用レジスターの値は，呼び出し元へ影響しないようにしている．サブルーチンでの処理の前にスタックへ格納 (PUSH) し，処理の後に取り出し (POP) で元に戻している．

とすれば良い．こうすれば，呼び出し元とサブルーチン間のデータの受け渡しは汎用レジスターを通しておこなわれ，その値の変化は必要最小限に抑えられる．他に影響が無いので，サブルーチンの独立性は高くなる．

3.9 [例題 9] ラベルを 2 重に付ける方法

教科書の例題 6(p.97)〜8(p.101) は，いずれも与えられたデータの最大値を求めるプログラムであった．これらの場合，最大値を求めたいデータの数列とその数が与えられていた．データ数を元に，数列を読みしと比較を繰り返すことにより最大値を探索した．ここでは，データ数が与えられていない場合，ラベルを 2 重につけてデータの終わりを示す．

教科書のプログラムの内容は，

- ラベル DATA が示すアドレスからデータが格納されている．
- アセンブラ命令 DS を上手に使うことにより，データの終わりのアドレスは，ラベル LAST-1 で示している．
- アドレス DATA〜LAST-1 に格納されている数列を合計して，ラベル SUM に格納する．

である．このプログラム例で学習することは，最終データがあるアドレスにラベル名をつけることである．それは，プログラム中で示しているように

```
DATA  DC   1,5,6,8,9
LAST  DS   0       ; 数列の最終アドレス+1
```

とするのである．こうすると数列の先頭のアドレスは DATA で，最終アドレスは LAST-1 で示すことができる．

3.10 [例題 10] 数値データを文字データに変換

3.10.1 文字コードに変換

マクロ命令の OUT を用いて，出力装置に数値を打ち出す場合，数値を文字データに変換しなくてはならない．なぜならば，OUT が出力できるのは文字に限られるからである．たとえば，整数の $(2345)_{10}$ を出力装置で打ち出す場合，#0032，#0034，#0034，#0035 と言うデータが必要である．これは，文字コードの規格で，数字 (0〜9) が文字コードの#0030〜#0039 に割り当てられているからである．

そのような理由で，整数 (0〜9) の値に#0030 を加えれば文字コードに変換できる．いろいろな方法が有るが，教科書では次のようにしている．

```
        OR    GR3,MOJI      ;#0030 を加算     MOJI=#0030
```

3.10.2 わり算

教科書のプログラムでは，割り算の演算も必要で，商と余りが計算できなくてはならない．それは何回も使うのでサブルーチンにしている．このサブルーチンのわり算の方法は簡単で，

- 被除数から除数の引き算を行う．引いたあまりが負になる場合は，引き算をやめる．この引き算の回数が商となる．
- 引き算の処理を行った残りが余りである．

としている．これは，除数も被除数も正と仮定しているためこのようなことができる．負の数を考えるともう少し複雑なことを考えなくてはならない．

3.11 [例題 11] 文字データを数値データに変換

マクロ命令の IN を使って，キーボードから数値を読み込みたい．問題は，この命令で読み込まれるのは文字であるということである．整数を入力しても，それは文字として処理され，指定されたメモリーに人文字ずつ格納される．すなわち，-102 とキーボードから入力された場合，メモリーには文字コード表に従い #002D，#0031，#0030，#0032 と格納される．このキーボードから入力された整数 (-102) を演算に利用するためには，文字コード化された整数を数値に変換する必要がある．

この文字コードと整数の対応を見ると，文字コードの最後の 4 ビットが整数に対応していることが分かる．この 4 ビットを取り出すのは簡単で，マスク処理を行えば良い．教科書では，汎用レジスター GR3 に文字を入れて，それをマスク処理して，整数に変換している．つぎのようにである．

```
AND    GR3,=#000F    ; マスク処理　最後の 4 ビットを取り出して，整数化
```

最後に，先頭の負号の確認を行う必要がある．先頭の文字が負号ならば，ラベル MINUS へ処理が移すために，教科書では次のようにしている．

```
CPA    GR3,='-'      ; 先頭の桁の処理　先頭を GR3 に入れて'-' と比較
JZE    MINUS         ; マイナスの時は分岐      GR3 が'-' ならば MINUS へ
```

そして，マイナスの場合は絶対値の処理が必要で，-1 倍すればよい．それは 2 の補数にすればよく，ビット反転と 1 加算すると実現できる．1 との排他的論理和 (XOR) を計算することにより，ビット反転はできる．教科書のプログラムでは，-1 倍は次のようにしている．

```
MINUS  XOR   GR0,=#FFFF  ; ビット反転
       ADDA  GR0,=1       ;+1 加算
```

3.11.1 かけ算

教科書のプログラムでは，かけ算の演算も必要である．それは何回も使うのでサブルーチンにしている．このサブルーチンのかけ算の方法は簡単で，

- ループを使って，桁の重みをその桁の値だけ加算している．

としている．

4 勉強方法

4.1 試験範囲

第 8〜15 回までの講義内容が試験範囲である．教科書の p.28-119 である．これは，教科書の 3-5 章に対応する．第 7 回の講義のシミュレーターは試験範囲外とする．

また，以下の資料は，学年末試験の問題の一部として，与えられるので暗記する必要はない．

- 教科書 p.213 の命令語の構成
- 教科書 p.209 の文字の符号表

命令語の構成の表が試験では与えられるので，機械語命令語を丸暗記する必要はないが多少の英語は分かって欲しい．アセンブラ命令とマクロ命令はこの表に載っていないので，覚える必要がある．数個なのでそれくらいは覚えられるだろう．

4.2 学習順序

次のような手順で勉強すれば効率が良いでしょう．

- 教科書 p.213 の命令語の構成の表を見ながら，その動作を理解する．
- このプリントと教科書を見ながら，ここで学習した例題に示している CASL II の基本的なテクニックを理解する．
- 理解できないところは，授業中に配布したプリントを参考にして考える．プリントを紛失した者は，web から入手せよ．
- 基本的なテクニックが理解できたならば，教科書の例題のプログラムの内容を理解する．以下の要領で理解すれば，上達が早い．これらについても，授業中に配布したプリントに，プログラム毎に書いてあるので参考にすると良いであろう．
  - まずは，プログラムの構造 (メインルーチン，サブルーチン，データ) がどうなっているか理解する．
  - 次に，プログラムの全体の流れを理解する．フローチャートを見よ．
  - 最後に 1 行毎に，その内容を理解する．

超重要

2003 年度の学年末試験と 2004 年度の後期中間試験，学年末試験の問題を解くこと．解答付きで web に載せている．ほぼ同様の出題をするつもりである．