CASL IIのプログラム例(その1)

山本昌志*

2006年2月2日

1　今後の学習

今後は，CASL IIを通して，アセンブラーのプログラムの書き方を学習する．教科書 [1] に沿って進めるが，ある人にとっては簡単すぎるであろう．また，基本情報技術者試験を受ける人にとっては，内容は不足している．そのため，この程度の内容が理解できる人は，講義とは別に勝手に進んで欲しい．

2　プログラムの見方

以前述べたように，プログラムは命令とデータから構成される．高級言語の場合，プログラムの大部分は命令である．データは外部のファイルから呼び出すことが多いので，ソースプログラムには書かれないことが多い．というか，あちこちに書いて，どうなっているのか分からないことも多い．それに対して，機械語やアセンブラのソースプログラムでは，命令部とデータ部を明確にするのが良い．

アセンブラのプログラムを見る場合，まず最初に，どの部分が命令で，どの部分がデータかを見分けなくてはならない．また，命令部はメインルーチンとサブルーチンをも見分けなくてはならない．これらを，見分けるのは簡単である．慣れればすぐにわかるが，大体の目安は，以下のとおりである．

- メインルーチンは，START命令で指定されたアドレスからRET命令までである．
- サブルーチンは，メインルーチン同様RET命令で終わっている．始まりはSTART命令が有ったり無かったりであるが，必ずラベルはある．メインルーチンと異なるのは，それがメインルーチンあるいは他のサブルーチンからCALL命令で呼び出されていることである．実際のところ，メインルーチンもOSからCALL命令で呼び出されているため，諸君が書くプログラムには無いのである．ハードウェアーにとっては，メインルーチンもサブルーチンも区別していないのである．
- データ部は，DCあるいはDS命令が書かれている行である．

プログラムを構成するこれらの3つの要素は，それぞれ一塊にかかれるのが普通である．一塊に書かないようにもできるが，それは非常にわかりにくいプログラムとなり，絶対に避けるべきである．少しでも

*国立秋田工業高等専門学校　電気工学科

経験のあるプログラマーならば，これらの要素は区別して分かりやすく書いている．センスのない人ほど，複雑でわかりにくいプログラムを書く．

> ソースプログラムを見たり書いたりする場合，メインルーチンとサブルーチン，データ部に分ることから始めよ．

3 [例題 1] 加算

教科書 [1] の List5-1 のプログラムを例にして，加算方法について説明する．

3.1 高級言語との違い

高級言語の場合，加算は，

```
wa=a+b;
```

のように書けばよい．数式と同じように書けば，`a+b` を実行し，その結果を変数 `wa` に格納する．

しかし，アセンブラーでは，こんなに簡単ではない．加算は CPU で行うが，そのためのデータは CPU が持っているメモリーであるレジスターを使って計算を行う．そのため，次のような手順が必要である．

1. 計算の対象のデータを，メモリーから，レジスターにロードする．
2. 計算を行う．計算結果は，レジスターに蓄えられる．
3. レジスターに蓄えられた計算結果をメモリーにストアーする．

全ての計算は，レジスターを通して行われるので，このような手間が必要なのである．これは，コンピューターのハードウェアーがそうなっているからである．アセンブラー言語は，コンピューターのハードウェアーを反映しているのである．アセンブラー言語でプログラムをする場合，ハードウェアーの思い浮かべれば，高級言語との違いが明確になる．

実際，高級言語ではコンパイラー[1]が，レジスターの処理とかの機械語のプログラムに直している．これは，同じハードウェアーで計算するので，やはり，高級言語と言えども最終的にはレジスターを使っているのである．

3.2 プログラムの構造

まず，リストを見て，プログラムが図 3 の構造になっていることを理解しなくてはならない．プログラムは，訳の分からない呪文が連なっているのではなく，ちゃんと整理すれば理解できる．まずは，これが第一歩である．

[1] 高級言語のソースプログラムを機械語に変換するプログラム

図 1: 教科書の List5-1 のプログラムの構造.

3.3 フローチャート

このプログラムのフローチャートを図 2 に示す．このプログラムは簡単で，水が上から下に流れるように命令部が順番に実行されるだけである．このように，書かれた順にプログラムが実行される構造を「順次」と言う.

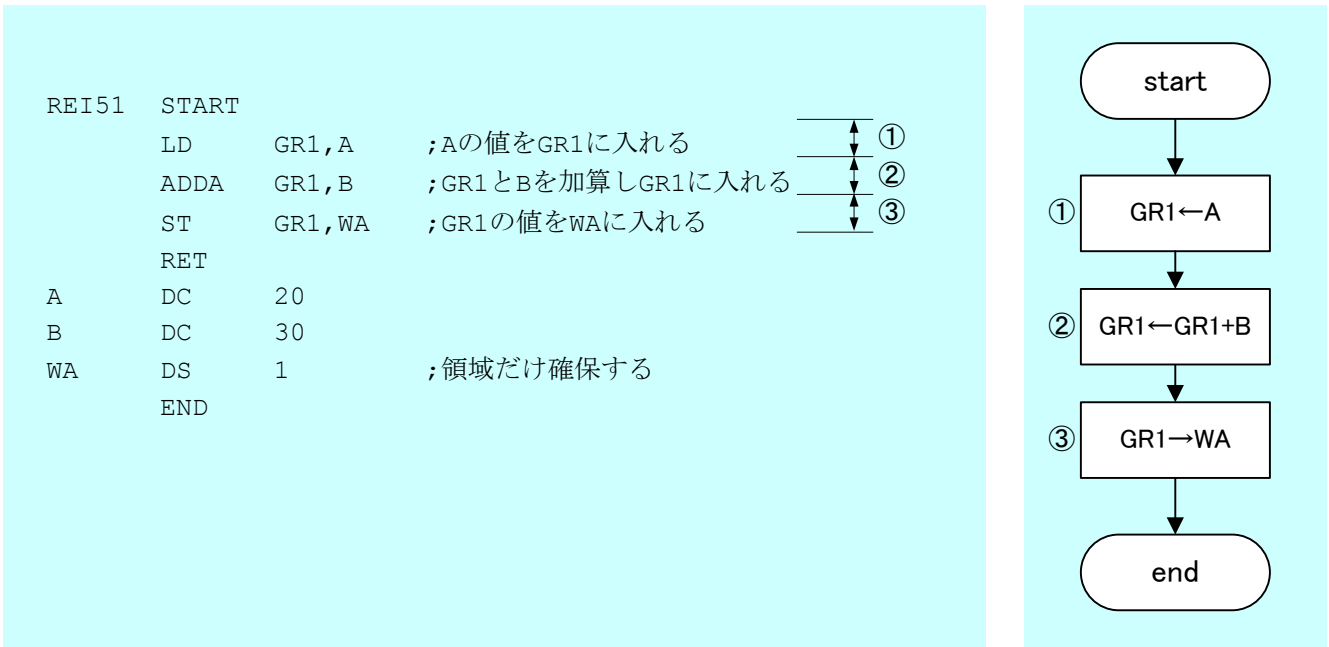

図 2: 教科書の List5-1 のプログラムのフローチャート.

4 [例題 2] 加算と条件分岐

教科書の List5-2 のプログラムを例にして，条件分岐について説明する．

4.1 高級言語との違い

高級言語の場合，条件分岐は簡単に実装できる．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 のよう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 (JMI，JNZ，JZE，JUMP，JPL，JOV) を使い，実行する文を選択する．
3. ジャンプ先は，ラベルで指定する．

4.2 プログラムの構造

まず，リストをみて，プログラムが図 3 の構造になっていることを理解しなくてはならない．

図 3: 教科書の List5-2 のプログラムの構造.

4.3 フローチャート

このプログラムのフローチャートを図 4 を示す．このプログラムは，[例題 1] とは異なり，プログラムの実行が条件に従い分岐する．このようなプログラムの構造を「選択」という．ここでは，この選択がフラグレジスターの値を制御値としてジャンプ命令で実装されていることを理解しなくてはならない．

それから，出力命令 OUT も理解しなくてはならない．

- OUT 命令の最初のオペランドは，出力したい文字が格納されている先頭アドレスである．
- 2 番目のオペランドは，出力する文字数が格納されているアドレスである．これがないと，何文字出力するか，コンピューターは分からない．

```
REI52   START
        LD      GR1,A       ;Aの値をGR1へ入れる        ①
        ADDA    GR1,B       ;GR1=GR1+B                ②
        JOV     L1          ;OFが1ならばL1へ          ③
        JUMP    L2          ;無条件でL2へ             ④
L1      OUT     BUFF,LEN    ;'OVER'を表示             ⑤
L2      ST      GR1,WA      ;GR1の値をWAへ入れる      ⑥
        RET
A       DC      20000
B       DC      30000
WA      DS      1
BUFF    DC      'OVER'      ;表示する文字列
LEN     DC      4           ;表示する文字列の長さ
        END
```

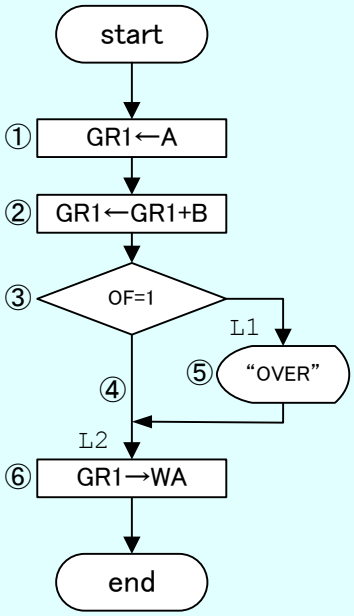

図 4: 教科書の List5-2 のプログラムのフローチャート.

5 [例題 3] マスク処理と条件分岐

教科書の List5-3 のプログラムを例にして，マスク処理と条件分岐について説明する.

5.1 マスク処理

5.1.1 教科書の例

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.

ここでは，次のようにしている．$(1452)_{10} = (0000010110101100)_2$ なので

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

としている．仮に，ラベル A の値が $(1453)_3$ としたら，

```
          0000010110101101
AND       0000000000000001
----------------------------
          0000000000000001
```

となる．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．

5.1.2 特定のビットパターンのマスクの方法

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である．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，否か比較する．

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      ; 定数の定義
```

同じ様なことが，ブール代数の双対の原理[2]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．

[2] 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

5.2 プログラムの構造

プログラムの構造については，先に説明した．ほとんど同じである．以下について，答えよ．

- メインルーチンとデータ領域の行番号を答えよ．
- ラベルと命令コード，オペランド，コメント文はどれか?

5.3 フローチャート

このプログラムのフローチャートを図 5 に示す..

```
REI53   START
        LD      GR0,A       ;GR0にAの値を入れる          ①
        AND     GR0,MASK    ;マスク処理                  ②
        JNZ     KISUU       ;結果が0でないときKISUU      ③
        OUT     GU,LEN      ;'GUUSUU'の表示              ④
        JUMP    FIN         ;終了へ                      ⑤
KISUU   OUT     KI,LEN      ;'KISUU 'の表示              ⑥
FIN     RET                 ;プログラム終了
A       DC      1452
MASK    DC      #0001
KI      DC      'KISUU '
GU      DC      'GUUSUU'
LEN     DC      6           ;表示する文字数
        END
```

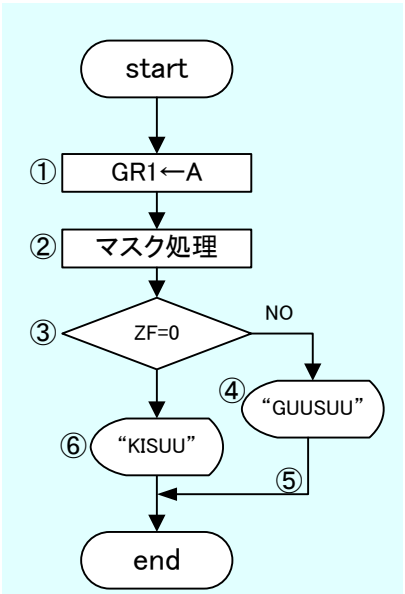

図 5: 教科書の List5-3 のプログラムのフローチャート．

6 練習問題

以下の練習問題は，レポートとして提出すること．

[**問 1**] 加算 (I)

- ラベル名 AA が示すメモリーの領域に $(10)_{10}$，BB が示す領域に $(30)_{10}$ の値を格納する．
- それぞれを加算した結果をラベル名 WA が示すメモリーの領域に格納する．

[**問 2**] 加算 (II)

- ラベル名 AA が示すメモリーの領域に $(FF00)_{16}$，BB が示す領域に $(00AB)_{16}$ の値を格納する．
- それぞれを加算した結果をラベル名 WA が示すメモリーの領域に格納する．

[**問 3**] 加算 (III)

- ラベル名 AA が示すメモリーの領域に $(-50)_{10}$，BB が示す領域に $(10FF)_{16}$ の値を格納する．
- それぞれを加算した結果をラベル名 WA が示すメモリーの領域に格納する．

[**問 4**] 減算 (I)

- ラベル名 AA が示すメモリーの領域に $(10)_{10}$，BB が示す領域に $(30)_{10}$ の値を格納する．
- $(10)_{10} - (30)_{10}$ の計算結果をラベル名 SA が示すメモリーの領域に格納する．

[**問 5**] 減算 (II)

- ラベル名 AA が示すメモリーの領域に $(-50)_{10}$，BB が示す領域に $(10FF)_{16}$ の値を格納する．
- $(-50)_{10} - (10FF)_{16}$ の計算結果をラベル名 SA が示すメモリーの領域に格納する．

[**問 6**] 減算と表示

- ラベル名 AA が示すメモリーの領域に $(-50)_{10}$，BB が示す領域に $(10FF)_{16}$ の値を格納する．
- $(-50)_{10} - (10FF)_{16}$ の計算結果をラベル名 SA が示すメモリーの領域に格納する．
- SA が負の値の場合，MINUS と表示する．正の場合，PLUS と表示する．

[**問 7**] 特定ビットの検査

- ラベル名 DATA が示すメモリーの領域に $(FFAA)_{16}$ の値を格納する．
- マスクを利用して，第 15 ビット (符号ビット) を検査する．
- 第 15 ビットが 1 の値の場合，MINUS と表示する．0 の場合，PLUS と表示する．

[**問 8**] 複数のビットの検査

- ラベル名 DATA が示すメモリーの領域に $(A0B9)_{16}$ の値を格納する．
- マスクを利用して，第 15 と第 0 ビットを検査する．
- 第 15 ビットが 1，第 0 ビットが 0 の場合 OK と表示する．それ以外の場合，NG と表示する．

6.1 レポート提出要領

提出方法は，次の通りとする．

期限　　2 月 17 日 (金) PM 1:00
用紙　　A4
提出場所　山本研究室の入口のポスト
表紙　　表紙を 1 枚つけて，以下の項目を分かりやすく記述すること．
　　授業科目名「電子計算機」
　　課題名「課題　プログラム練習 (その 1)」
　　3E　学籍番号　氏名
　　提出日
内容　　2 ページ以降に問いに対する答えを分かりやすく記述すること．

参考文献

[1] 東田幸樹, 山本芳人, 広瀬啓雄. アセンブラ言語 CASL II. 工学図書 (株), 2002 年.