アセンブラ命令 (非実行文)

山本昌志*

2005 年 12 月 16 日

1　前回の復習と本日の学習

1.1　前回の復習

前回の講義では，CASL II のシミュレーターである WCASL II [1] の使い方を学習した．今後，CASL II のプログラムを学習するときに役立てて欲しい．

中間試験前になるが前々回の講義では，CASL II の命令の種類とプログラムの書き方について学習した．プログラムの書き方は図 1 のとおりで，4 つの記述欄があり，それぞれは 1 つ以上の空白で区切る．また，セミコロン ; を書けば，行のそれ以降は注釈文 (コメント文は) となりアセンブラーは無視をする．

さらに，命令は以下の 3 種類に分けられることも学習した．

アセンブラ命令　ソースプログラムを機械語に変換するアセンブラーに対しての命令で，特定のビットパターンに変換されない．また，この命令は CPU を動作させることはない．

機械語命令　CPU の実際の動作を指示する命令で，特定のビットパターンに変換される．この命令と機械語は 1 対 1 の対応がある．

マクロ命令　複数の機械語命令を集めて，一つの名前を付けた命令である．これは特定の機械語に変換されるが，ひとつの命令は多くの機械語になる．

1.2　本日の学習内容

教科書の p.28-35 のアセンブラ命令を学習する．ゴールは以下のとおりである．

- アセンブラ命令の役割が分かる．
- CASL II のアセンブラ命令 (`START`，`END`，`DC`，`DS`) の使い方が分かる．

である．

*国立秋田工業高等専門学校　電気工学科

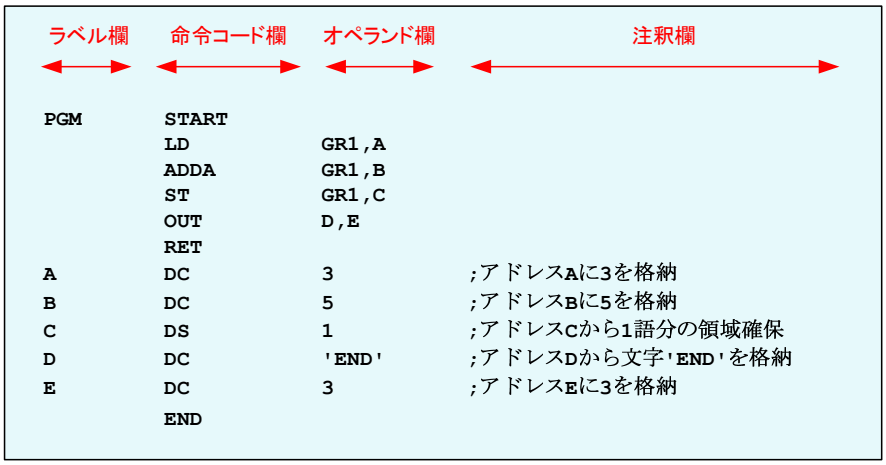

図 1: CASL II のプログラムの書き方

2　アセンブラ命令と機械語命令の違い

以前の講義で図 1 のプログラムを学習した．このプログラムを実行する過程を考えよう．テキストエディター[1]を使って，このプログラムを書き上げると，次の操作を行い，プログラムを実行させる．

1. アセンブラーをつかって，図 1 のソースプログラムを機械語に直したファイルを作成する．
2. 機械語のファイルを実行させる．
   - 機械語のファイルを COMET II のメモリーに格納する．
   - OS が実行する命令が書かれた先頭番地をコールする．
   - 命令に従いプログラムが実行される．

これらプログラムのアセンブルと実行の操作で，アセンブラ命令と機械語命令 (マクロ命令も) は，指示する相手が決定的に異なる．

- アセンブラ命令は，ソースプログラムをアセンブラーがアセンブルして実行ファイルを作成するときに，アセンブルの方法を示したものである．アセンブラ命令は，このアセンブルするときのアセンブラーに対して指示を行う．
- 機械語命令はプログラム実行時に，CPU に対して指示を行う．

教科書では，アセンブラ命令のある文を非実行文と書いてある．プログラムを実行させた場合，アセンブラ命令の行は実行されないからそのように呼ばれるのである．

プログラマーは，アセンブラ命令と機械語命令の違いをしっかりと認識しなくてはならない．CASL II には，アセンブラ命令は，次の 4 個しかない．以降，それぞれについて説明する．

[1]テキストファイルを編集するプログラム．Windows のワードパッドなど．

START プログラムの実行開始行を示す.

END プログラムの記述の終わりを示す.

DC メモリーを確保して，初期化を行う.

DS メモリーを確保する.

3 プログラムの開始と終了

3.1 プログラムの始まり (START)

書式

ラベル欄	命令コード欄	オペランド欄
label	START	[実行開始番地]

- プログラムの先頭は，START 命令で始まらなくてはならない.
- ラベルは，必要不可欠である.
- 実行開始番地は無くても良い．絶対番地を書くことは稀で，通常，最初に実行する命令が書かれている行のラベル名を書く.
- 非実行文なので，フラグレジスタは変化しない (関係ない).

この命令の役割は，プログラムの実行開始番地 (アドレス) をアセンブラーに対して指示することである．プログラムの先頭に必ず書く必要があり，これが無いと，どこからプログラムを実行するか分からない．プログラム実行時に，この START 命令が示すアドレスが最初の PR(プログラムレジスタ) の値になる．これが実行開始番地である.

リスト 1 のプログラムで START 命令の役割を考える．このプログラムをアセンブラーが機械語に変換すると，図 2 のようになる．START と END 命令はアセンブラ命令であるため，マシン語に変換されない．他のアセンブラ命令，DC や DS は，それが示すデータに変換される.

リスト 1: CASL II のプログラム例．3+5 を計算する

```
1 PGM     START   BEGIN
2 BEGIN   LD      GR1 , A
3         ADDA    GR1 , B
4         ST      GR1 , C
5         RET
6 A       DC      3
7 B       DC      5
8 C       DS      1
9         END
```

START 命令はマシン語に変換されないが，プログラマーはアセンブラーに，このプログラムはラベル BEGIN から実行すると言うことを伝えなくてはならない．ラベル BEGIN は，最初に実行する命令

```
    LD GR1,A
```

の先頭番地が格納されているアドレスを示す．そのアドレスがプログラム実行開始時にプログラムレジスター PR にセットされる．それを確実にするために，START 命令がある．この命令で，最初に実行させる命令を示すのである．CPU は，その番地に書かれた 0 と 1 の集まりは命令と解釈する．

START 命令など無くして，最初の行から実行するように約束することも出来るであろう．こうすると，いつも先頭の行から実行することになり，実際プログラムを書く場合不便なことがある．START 命令が有った方が，便利なのである．

もし，オペランド欄に記述が無い場合，START 命令の次の行からプログラムの実行は開始することになっている．START 命令のラベルは，そのプログラム自身で参照されることはないが，絶対に必要である．ほかのプログラム，たとえば OS がプログラムの実行を指示する場合，このラベルが使われる．そしてこのラベルが示すアドレスは，このプログラムの実行開始番地になる．したがって，図 1 のラベル PGM と BEGIN は同じアドレス#0020 である．

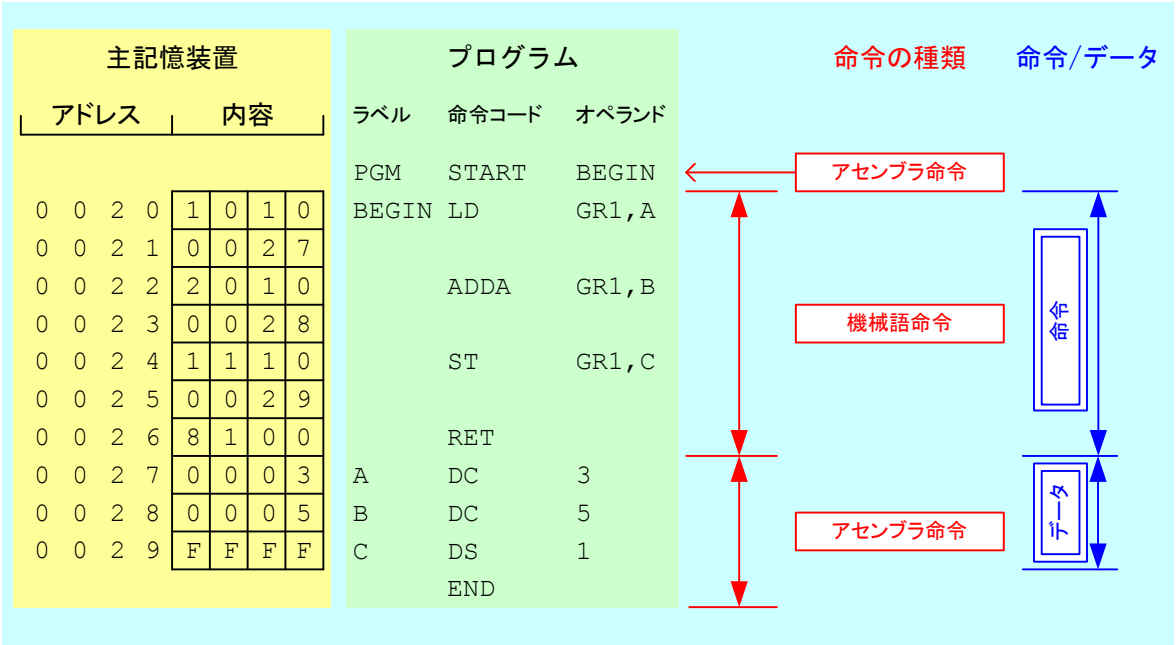

図 2: メモリーの内容と命令の種類

3.2 プログラムの終わり (END)

書式

ラベル欄	命令コード 欄	オペランド 欄
	END	

- プログラムの最後に，必ず END 命令を書かなくてはならない．
- ラベル欄とオペランド欄は，書いてはならない．
- 非実行文なので，フラグレジスタは変化しない (関係ない)．

この命令の役割は，プログラムの終わりをアセンブラーに対して知らせることである．プログラムの最後に必ず書く必要があり，これが無いと，どこでプログラムが終わったのか分からない．プログラムの実行が終わったところではなく，プログラマーが記述したソースコードの終わりを示す．

END 文はラベルをつけることはできない．この命令は，主記憶装置に格納されないので，対応するアドレスが無いからである．また，処理の対象となるオペランドも無いので，書くことはできない．しかし，END 文の後に注釈は許される．アセンブラーは，これは無視する[2]．

4 定数の定義 (DC)

4.1 内容

以前から述べているように，プログラムは命令とデータから構成させれる．このうちデータを書き表すために，DC が使われる．これは Define Constant の略で，定数を定義するように思えるが，実際はメモリーを確保して初期化しているだけである．したがって，定数と言いながら，メモリーの内容は書き換え可能である．

4.2 書式と例

DC:Define Constant

書式

ラベル欄	命令コード欄	オペランド欄
[label]	DC	n
[label]	DC	#h
[label]	DC	’文字列’
[label]	DC	ラベル名
[label]	DC	定数[,定数,定数,]

- ラベル名は，確保された領域の先頭のアドレスである．
- ラベルは無くても良いが，データへのアクセスが煩雑になる．通常，ラベルを付けて，それを使ってデータにアクセスする．
- オペランド欄に書かれるデータは，以下の場合に分けられる．

[2] END 命令に限らず，どこにある注釈でもアセンブラーは無視する．注釈はプログラマーのためのものであり，アセンブラーはいっさい関知しない．

- 10 進数の整数 (n)
  - 10 進数整数のデータを定義する．CASL II の場合，整数は 16 ビットで表されるため，その範囲は-32768〜32767 までである．これを超えた場合，その下位 16 ビットがビットパターンに変換される．
- 4 桁の 16 進数の正の整数 (h)
  - 16 進数整数のデータを定義する．その範囲は#0000〜#FFFF までである．
- 文字列
  - 文字列のデータを定義する．CASL では 1 文字を 16 ビット (1 ワード) で表現するが，JIS X 0201 では 8 ビットで表現する．このことより，上位 8 ビットは 0 とし，下位 8 ビットで表現することになっている．以前学習したとおりである．
  - アポストロフィ「’」をデータとして使いたい場合は，それを 2 つ続けて「’’」のように書く．そうすると，1ヶのアポストロフィがデータとして定義される．
  - ラベルは，第一文字目のデータが格納されたアドレスを示す．
- ラベル名
  - アドレス定数とよばれるもので，記号番地であるラベル名をアドレスの絶対番地メモリーに格納できる．指定されたラベル名の絶対番地がメモリーに格納される．
- カンマ区切りの複数のデータ
  - カンマで区切って，いくつでも定数を書くことができる．

- 非実行文なので，フラグレジスタは変化しない (関係ない)．

実際の例を以下に示す．特に説明するまでも無いだろ．

```
AA  DC  100
BB  DC  -3
CC  DC  #0027
DD  DC  'AT<&'
EE  DC  AA
FF  DC  10,20,'AB',#FFFF
```

5 領域の確保 (DS)

5.1 内容

領域の確保の命令で，プログラムの実行に必要なメモリーの領域を確保する．実際のプログラムでは，計算途中や結果のデータを格納するために，記憶領域が必要となる．

5.2 書式と例

DS:Define Storage

書式

ラベル欄	命令コード 欄	オペランド 欄
[label]	DS	n

- ラベルは，無くても良いが，メモリー領域へのアクセスが複雑になるので，通常は付ける．
- n は，10 進数の整数である $(0 \leqq n)$．n 語分の領域を確保する．
- ラベル名は，確保された領域の先頭のアドレスである．
- 非実行文なので，フラグレジスタは変化しない (関係ない)．

以下に実際のプログラムで使われる例を示す．ここで，おもしろ使い方をしてるのが，ラベル BB の行である．メモリーはひとつも確保されないが，ラベル名がある．この場合，BB が示すアドレスは CC が示すアドレスと同一になる．

```
AA  DS  2
BB  DS  0
CC  DC  'AB'
```

参考文献

[1] 渡辺博芳. WCASL-II. http://www.geocities.jp/chokyumei/tankoubon/hanoi.html.