これまでの復習 (後期中間試験に向けて)

山本昌志*

2005 年 11 月 25 日

もっとも手っ取り早く点数がとれる勉強は，過去問を理解することである．最低限，昨年と一昨年の過去問が解けるようになること．昨年までは 50 分の年間授業であったため，試験範囲は昨年までの前期末試験の途中までなので注意すべし．過去問は，私の web ページ (http://www.akita-nct.jp/~yamamoto/) からたどれば見つかる．

1 CASL II とは

- コンピューター内部では，データと命令は 0 と 1 の 2 進数で表現されます．たとえば，加算命令 (足し算) は，00100000000100000000000000001010 です (教科書 p.1)．これを機械語といいます．
- これは，覚えるのも大変なので，この命令を人間にわかり易くする工夫が考えられました．0 と 1 の機械語の代わりに，ADDA GR1, ADDRES と，表記するようにしたのです (教科書 p.1)．これがアセンブラ言語です．
- 実際のコンピューターの動作は機械語なので，アセンブラ言語は，アセンブラーと言うプログラムで，機械語に変換します．
- 高級言語，たとえば FORTRAN とアセンブラ言語には，大きな違いがあります．高級言語の 1 個の命令をコンパイルしてマシン語に変換すると，それは数多くのマシン語から構成されます．一方，アセンブラ言語をアセンブルすると，1 個のマシン語になります．即ち，アセンブラ言語はマシン語と 1 対 1 の対応があります．
- アセンブラ言語は CPU の動作を指示するものとも言えます．したがって，CPU の種類によりそのアセンブラ言語は異なります．
- 基本情報技術者試験でも，アセンブラ言語があります．その場合，CPU 毎に試験をしていたのでは，大変です．そこで，仮想のアセンブラ言語，CASL II が考えられました．このアセンブラ言語が動作する仮想のハードウェアーを COMET II といいます．

*国立秋田工業高等専門学校　電気工学科

2　チューリング機械とノイマン型コンピューター

- チューリング機械は，図 1 のような構造をしています．そして，その動作は，次の通りです．
  - 書き換え可能な無限に長いテープと，オートマトンと言われる移動可能な機械からできている．
  - テープには，いろいろな記号が書かれている．
  - オートマトンには，テープの内容を読み書き可能なヘッドと内部状態を記憶する装置，テープの任意の位置に移動する装置から構成されている．
  - オートマトンの動作 (テープの読み書き) や移動は，今の場所のテープの記号と内部状態により決まる．

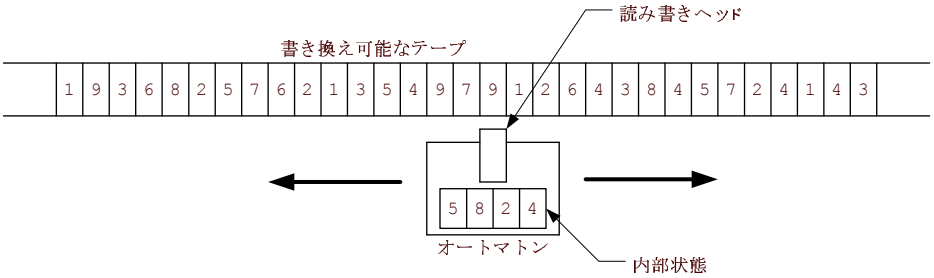


図 1: チューリング機械

- この単純なチューリング機械で，ほとんどあらゆる計算ができます．計算できない問題もあるようですが，これはここの講義のレベルを超えます．
- このチューリング機械を実際に実現させたものが，ノイマン型コンピューターです．その特徴は，以下の通りです．
  - 1 次元的に並んだメモリーがあり，そこにプログラム (命令) もデータも格納される．メモリーの内容は，自然数の番地で参照できる．
  - メモリーに格納されたプログラム (命令) とデータの見かけ上の区別はない．プログラムをデータとして見ることも，データをプログラムとしてみることもできる．

3　基数の変換 (2, 10, 16 進数)

- いろいろな数の表記方法があります．N 進数の場合，次のように N 個の底で数を表現します．

2 進数　$0, 1$
10 進数　$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
16 進数　$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}$

- 桁上がりは，2 進数の場合 1 の次で 10 に，10 進数の場合 9 の次で 10 に，16 進数の場合 F の次で 10 になります．
- 我々が通常用いている数の表現の意味は，次の通りです．数字の並ぶ順序が重要です．これを「位取り記数法」と言います．

$$(1905)_{10} = (1 \times 10^3 + 9 \times 10^2 + 0 \times 10^1 + 5 \times 10^0)_{10}$$

- 基数の変換 (2 → 10 進数)．通常の位取り記数法が理解できれば，簡単です．

$$\begin{aligned}(1101)_2 &= (1 \times 10^{11} + 1 \times 10^{10} + 0 \times 10^1 + 1 \times 10^0)_2 \\ &= (1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10} \qquad \leftarrow \text{普通はここから計算} \\ &= (8 + 4 + 0 + 1)_{10} \qquad \leftarrow \text{ここから計算しても良い} \\ &= (13)_{10}\end{aligned}$$

- 2 進数の各桁の 10 進数の値 (重み) を覚えておくと便利です．

$$1,\ 2,\ 4,\ 8,\ 16,\ 32,\ 64,\ 128,\ 256,\ 512,\ 1024,\ 2048,\ 4096$$

- 基数の変換 (10→2 進数)．2 で割った余りを並べます．変換方法の例を，以下に示します．$(19)_{10} = (10011)_2$，　$(2003)_{10} = (11111010011)_2$ です．

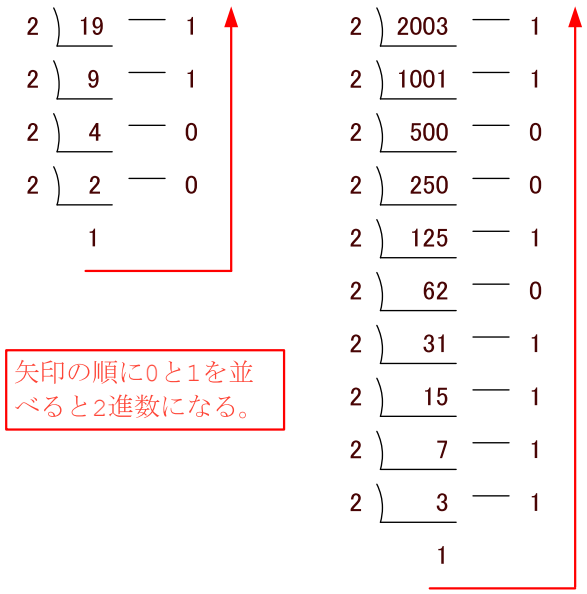
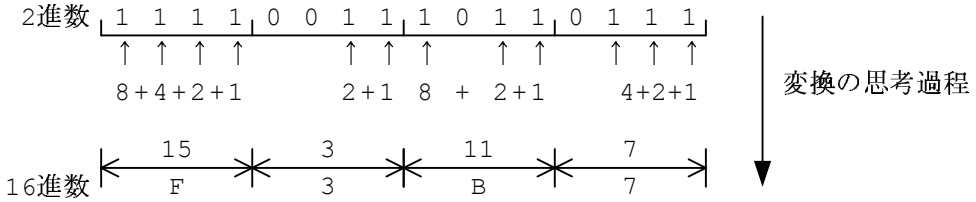


図 2: 10 進数から 2 進数への変換方法．

- 基数の変換 (16→10 進数)．これも，2 進数と同じです．

$$\begin{aligned}(376)_{16} &= (3\times 10^2 + 7\times 10^1 + 6\times 10^0)_{16}\\ &= (3\times 16^2 + 7\times 16^1 + 6\times 16^0)_{10}\\ &= (3\times 256 + 7\times 16 + 6\times 1)_{10}\\ &= (886)_{10}\end{aligned}$$

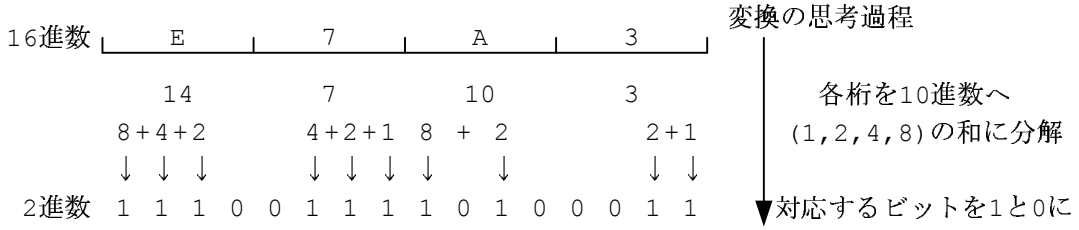
- 基数の変換 (2 → 16 進数)．2 進数の各桁を，最小桁から 4 桁ずつ区切り，それぞれを 16 進数に変換します．



$$(1111001110110111)_2 = (F3B7)_{16}$$

図 3: 2 進数から 16 進数への変換方法．

- 桁数が合わない場合は，先頭に必要なだけゼロを書き足して考えます．例えば，$(101100)_2 = (00101100)_2 = (2C)_{16}$ となります．
- 基数の変換 (16→2 進数)．16 進数の各桁を (1, 2, 4, 8) の和に展開して，それぞれのビットに対応させます．



$$(E7A3)_{16} = (1110011110100011)_2$$

図 4: 16 進数から 2 進数への変換方法．

- 基数の変換 (10→16 進数)．2 つの方法があります．

1. 一旦，2 進数へ変換した後，16 進数へ変換する．　← おすすめ

2. 16 で割って，その余りが各桁になる．

4 負の数の表現

- COMET II では，負の整数は 2 の補数で表現されます．メモリーの中に，16 ビットで格納されます．
- 負の数を 2 の補数で表現する手順は，以下の通りです．

  ① 負の数の絶対値を 2 進数で表現して，ビット反転する．

  ② +1 加算

  [例] $(-18)_{10}$ は，COMET II の内部，16 ビットの 2 の補数は，$(1111111111101110)_2$ と表されます (メモリーへの格納状態)．

$(-18)_{10}$	0000000000010010	←	18 の 2 進数表現 (16 ビット)
	1111111111101101	←	ビット反転
	1111111111101110	←	+1 加算

- 2 の補数を使うと，以下の有利な点があります．
  - 負の数の加算が通常の加算器で出来る．
  - 加算の場合の負の数，あるいは減算は，①2 の補数に変換して，②加算器による加算を行う．減算器を作るより，この方が回路が簡単になる．

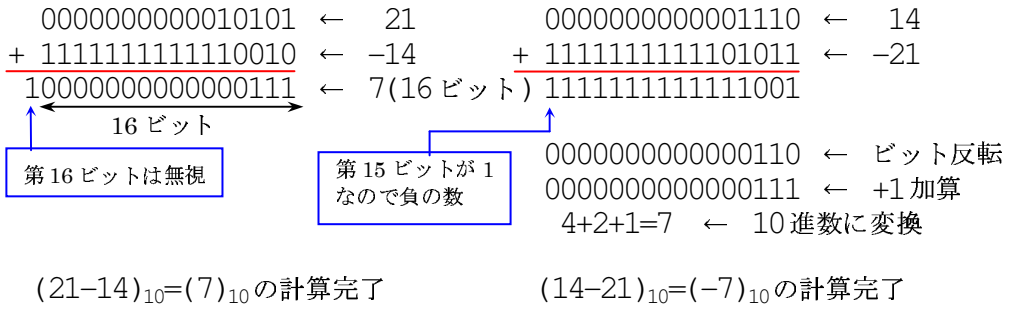


図 5: 補数を使った計算

- 2 の補数を求める手順 (①ビット反転 ②+1 加算) は，コンピューター内部表現では，$\times(-1)$ と同じです．
- COMET II の符号付き整数

– 正の数は 16 ビット 2 進数でそのままの表現です．一方，負の数は 2 の補数を使います．正か負かの判断は，最上位のビットで判断します．最上位の第 15 ビットが 0 ならば正，1 であれば負です．

– 最上位のビットが符号を表すため，絶対値は残りのビットで表すことになります．したがって，表現可能な整数は-32768〜32767 です．

$$\begin{array}{ll} \text{正の整数の最大値} & (0111111111111111)_2 = (2^{15}-1)_{10} = (32767)_{10} \\ \text{負の整数の絶対値の最大値} & (1000000000000000)_2 = (2^{15})_{10} = (32768)_{10} \end{array}$$

- COMET II の符号無し整数

  – 正の数は 16 ビット 2 進数でそのままの表現です．一方，負の数を表すことはできません．

  – 正の整数は，16 ビットのパターンが 2 進数と同じです．したがって，表現可能な整数は 0〜65535 です．

$$\begin{array}{ll} \text{最小値} & (0000000000000000)_2 = (0)_{10} \\ \text{最大値} & (1111111111111111)_2 = (2^{16}-1)_{10} = (65535)_{10} \end{array}$$

5 COMET II の文字の取り扱い

- 数値と異なり，文字にはそれぞれ，番号をつけて区別します (コード化)．文字とそれに対応する番号は，規格 JIS X0201 ラテン文字・片仮名用 8 単位符号で決まっています．

- この番号は，8 ビットなので，最大 256 文字しか使えません．数字とアルファベットと片仮名と記号を表すのであれば十分です．漢字は，使えません．

- COMET II の 1 ワード 16 ビットに対して，文字は 8 ビットしか使いません．COMET II では 1 ワードで 1 文字を表すため，16 ビットのうち上位 8 ビットは 0 として，下位 8 ビットで 1 文字分を表します．例えば，アルファベットの Yama を表す場合，Y は $(59)_{16}$，a は $(61)_{16}$，m は $(6D)_{16}$，という番号がついているので，COMET のメモリーには，次のように格納されます．ただし，アドレスの実際の割り当ては，OS が決めます．

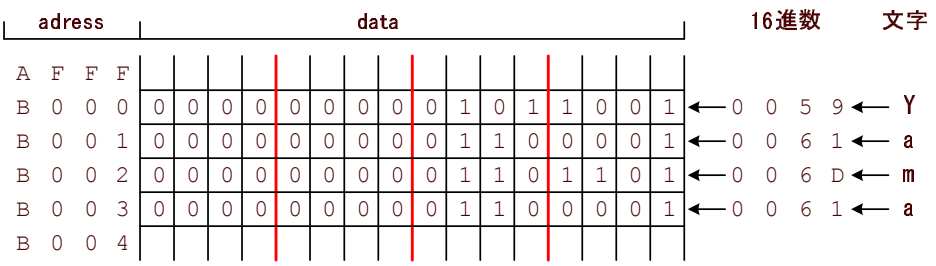


図 6: 文字列”Yama”のメモリーへの格納

- 数値と文字では，メモリーの中身は異なります．例えば，数値の $(9)_{10}$ と文字の”9”は，以下のようになります．文字の”9”は，JIS X0201 では，$(39)_{16}$ です (図 7).

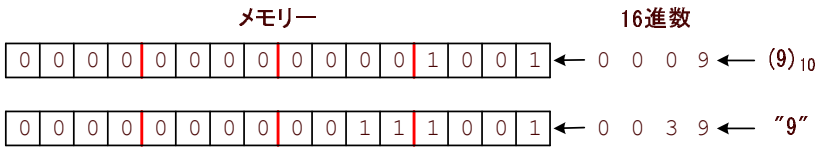


図 7: 数値の $(9)_{10}$ と文字”9”のメモリーへの格納

- メモリーの中身を見ると，それが数値なのか文字なのか，判断できません．命令毎に数値を扱うのか，文字を扱うのか決まっています.

6 主記憶装置とレジスタ

- COMET II では，16 ビットを 1 ワード (1 語) と言い，この単位でデータの処理をします.
- 主記憶装置 (メインメモリ) には，1 ワード (16 ビット) 毎にアドレスがついています．アドレスも 16 ビットです.
- コンピューターのプログラムは，データと命令から構成されます．この命令とデータは，実行時に主記憶装置 (メインメモリ) に格納されます.
- レジスタもデータなどを蓄えるので，主記憶装置同様，メモリの一種です．しかし，それぞれ，役割が異なります．主記憶装置は，いろいろなデータ (命令もデータの一種と考える) を蓄えるファイルキャビネットのようなものです．一方，レジスタは，実際に CPU がデータを加工するときに一時的に記憶する場所です.
- CPU と主記憶装置は，図 8 のような関係です．CPU は主記憶装置のアドレスを指定することにより，主記憶装置に格納されているデータを引き出します．そして，それはレジスタに記憶され，その中身に従い，処理されます．処理された結果ももちろん，レジスタに記憶されます．レジスタの中身を主記憶装置に戻すことにより，データの加工が完了します.

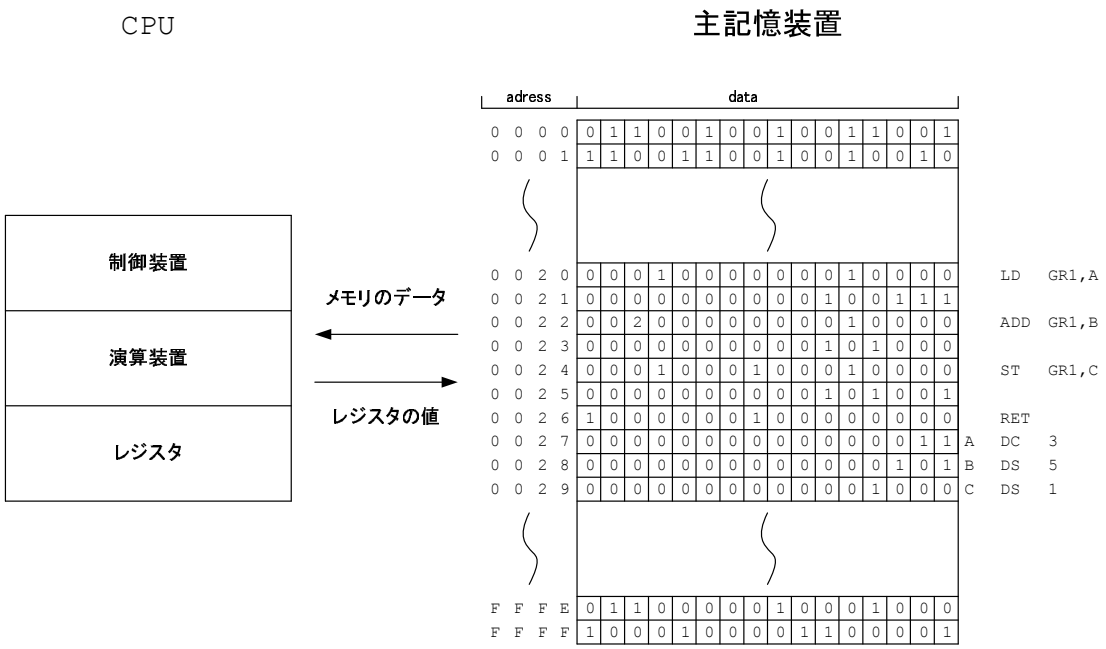


図 8: CPU と主記憶装置の関係

- COMET II のレジスタを表 1 にまとめておきます.

表 1: CASL II のレジスタ

記号	語源	日本語	機能
GR	General Register	汎用レジスタ	計算等に用いる．また GR1〜GR7 は指標レジスタとしても使われる．
SP	Stack Pointer	スタックポインタ	スタック領域の最上段のアドレスを保持する．
PR	Program Register	プログラムレジスタ	次に実行する命令のアドレスを保持する
FR	Flag Register	フラグレジスタ	演算結果の状態を保持する

  - 汎用レジスター
    - 計算等に主に用いられる．
    - 16 ビットのレジスターが 8 個 (GR0〜GR7) ある．
  - スタックポインター
    - スタック領域 (主記憶装置で CPU が記憶場所として使うことができる領域) の最上段のアドレスを格納している．

    * 16 ビットのレジスターが 1 個ある.
  - プログラムレジスター
    * 次に実行する命令のアドレスを格納している.
    * 16 ビットのレジスターが 1 個ある.
  - フラグレジスター
    * 計算結果などの状態を格納している.
    * 3 個の 1 ビットのレジスターがある.

      OF　計算結果がオーバーフローしたとき等，1 が設定される.
      SF　計算結果が負 (第 15 ビットが 1) のとき等に，1 が設定される.
      ZF　計算結果がゼロ (全てのビットが 0) とき等に，1 が設定される.

- 指標レジスターと言うものもあります.
  - 汎用レジスターの GR1〜GR7 が兼ねます．専用のハードウェアーは無いということです.
  - アドレスをオフセットするときに使います.

7　アセンブラ言語をマシン語に変換

- プログラムは，命令とデータから構成される．プログラムを実行するためには，アセンブラ言語を 0 と 1 のビットパターンのマシン語に変換する必要がある．命令とデータをビットパターンに変換するのである.
- リスト 1 は，3+5 を計算するプログラムである．これを，ビットパターンに変換すると，図 9 のようになる.
- 教科書の p.213 の命令語の構成に従って，命令のビットパターンに変換できる．これは，問題文に載せるので憶える必要は無い.
- この命令語の構成に書かれているオペランドを簡単にまとめると，次のようになる.

r	汎用レジスター	GR0〜GR7
r1	1 つの命令で 2 つの汎用レジスターを使うときの一方	GR0〜GR7
r2	もう一方の汎用レジスター	GR0〜GR7
adr	アドレスを示す.	レベル名が書かれることが多い.
x	アドレスをシフトするインデックスレジスタ.	GR1〜GR7

- 最初に機械語に変換される命令は，LD GR1,A である．その変換は，次のように行う.

1. LD という命令から，16 進数 4 桁の表示の最上位の桁は $(1)_{16}$ と分かる.
2. 次の桁は，LD には，$(0)_{16}$ か $(4)_{16}$ である．ここでは，LD r,adr,x のパターンとなっているので，次の桁は $(0)_{16}$ と分かる.

3. 次の桁は，汎用レジスターを示す．ここで使われている汎用レジスターは，GR1 なので，$(1)_{16}$ となる．
4. 次の桁は，インデックスレジスターを示す．インデックスレジスターは無いので，その桁は $(0)_{16}$ となる．

ここの命令は，LD r,adr,x のパターンであるので，命令語長は 2 語である．最初の 1 語は今示したとおり，$(1010)_{16}$ である．次の 1 語は，ラベル A のアドレスである．これは，プログラムが格納されるアドレスに依存する．ここでは，教科書 (p.17 の図 2.4) に沿って，$(A000)_{16}$ からプログラムは格納されるとすると，A のアドレスは $(A007)_{16}$ となる．これが第 2 語のビットパターンとなる．

リスト 1: CASL II のプログラム例．3+5 を系算する

```
1  PGM     START            ;プログラムの開始
2          LD      GR1,A    ;アドレス A の値を汎用レジスター GR1 に格納する
3          ADDA    GR1,B    ;アドレス B の値と GR1 の加算，結果は GR1 に格納する
4          ST      GR1,C    ;GR1 に格納されている値をアドレス C に入れる
5          RET              ;呼び出し元へ復帰する．
6  A       DC      3        ;メモリーを確保して，3 を格納する．
7  B       DC      5        ;メモリーを確保して，5 を格納する．
8  C       DS      1        ;メモリーを1ワード分確保する．
9          END              ;プログラムの終了
```

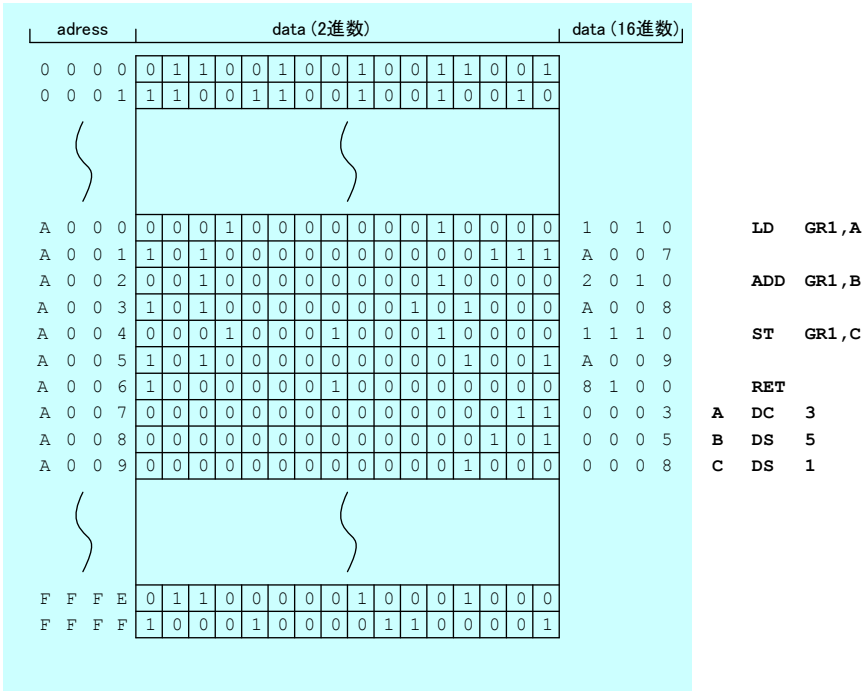


図 9: アセンブラ言語をマシン語へ変換

8　アセンブラ言語 (CASL IIの書き方)

- 人間が理解できるアセンブラ言語 (ここでは CASL II) のソースプログラムを，コンピューターが理解できるマシン語に直すプログラムをアセンブラーと言う．
- CASL II のアセンブラ言語の命令は，アセンブラ命令と機械語命令，マクロ命令に分けられる．
  - アセンブラ命令はアセンブラーに指示するためにある．CPU が実行する機械語に変換されない．ただし，データはビットパターンに変換される．
  - 機械語命令は，実際に CPU の動作を記述する．機械語命令は，CPU が動作するためのある特定のビットパターンに変換される．
  - マクロ命令は CPU が実行するビットパターンに変換されるが，機械語命令のような 1 対 1 の対応はない．多くの機械語命令を組み合わせて，その命令を実行する．
- CASL II のプログラムは，図 10 のように記述する．
  - ラベル欄に文字列は，アドレスを表す．オペランドとしてそれが使われると，マシン語では，その行のアドレスに変換される．
  - 命令コード欄には命令が書かれる．
  - オペランドには，命令コードが処理を行う対象を書く．
  - セミコロン (;) を書くと，それ以降から行の終わりまで，コメント (注釈) 文として解釈され，アセンブラーは無視する．

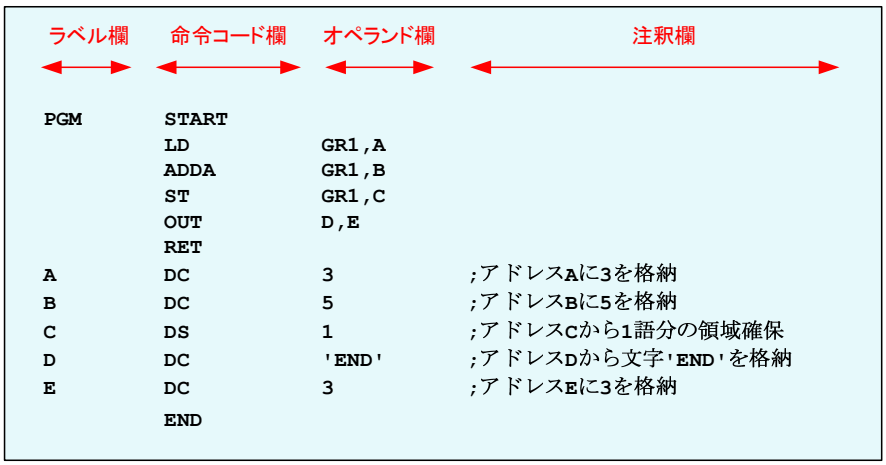


図 10: CASL II のプログラムの書き方