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-net.jp/~yamamoto/）からたとえば見つかる。

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、001000000010000000000000001010です（教科書p.1）。これを機械語といいます。

- これは、覚えれるもの大変なので、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、ADDRESS 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。

![チューリング機械](image)

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、これはここでの講義のレベルを超えるです。
- 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, A, B, C, D, E, 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) は、通常の位取り記数法が理解できれば、簡単です。

\[(1101)_2 = (1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0)_2\]

= \[(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10}\] 普通はここから計算

= \[(8 + 4 + 0 + 1)_{10}\] ここから計算しても良い

= \[(13)_{10}\]

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) 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。

\[(19)_{10} = (10011)_2, \quad (2003)_{10} = (11111010011)_2\]です。

![図2: 10進数から2進数への変換方法](attachment:binary_conversion_diagram.png)

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) これは、2 進数と同じです。

\[
(376)_{16} = (3 \times 10^2 + 7 \times 10^1 + 6 \times 10^0)_{16} \\
= (3 \times 16^2 + 7 \times 16^1 + 6 \times 16^0)_{10} \\
= (3 \times 256 + 7 \times 16 + 6 \times 1)_{10} \\
= (886)_{10}
\]

• 基数の変換 (2 進数→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。

\[
\begin{array}{c|cccc|cccc|cccc}
| 2 進数 | 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 |
| 8 + 4 + 2 + 1 & 2 + 1 & 8 + 2 + 1 & 4 + 2 + 1 |
\end{array}
\]

変換の思考過程

\[
\begin{array}{c|c|c|c|c}
| 16 進数 | 15 & F & 3 & 11 & E & 7 |
\end{array}
\]

\[
(1111001110110111)_{2} = (F3B7)_{16}
\]

図 3: 2 進数から 16 進数への変換方法。

• 株数が変わる場合は、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、それぞれのバイトに対応させます。

\[
\begin{array}{c|c|c|c|c}
| 16 進数 | E & 7 & A & 3 |
| 8 + 4 + 2 & 4 + 2 + 1 & 8 + 2 & 2 + 1 |
\end{array}
\]

変換の思考過程

各桁を 10 進数へ
(1, 2, 4, 8) の和に分解

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
| 2 進数 | 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 |
| 2 進数 | 0 & 0 & 0 & 1 | 0 | 1 & 0 & 0 | 0 | 1 |
\end{array}
\]

\[
(E7A3)_{16} = (1110011110000111)_{2}
\]

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，(111111111101110)2 と表されます（メモリーへの格納状態）。

\[
\begin{align*}
(-18)_{10} &\quad 0000000000010010 \leftarrow 18 の 2 進数表現 (16 ビット) \\
111111111101101 &\quad \leftarrow \text{ビット反転} \\
111111111101110 &\quad \leftarrow +1 \text{加算}
\end{align*}
\]

- 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。

\[
\begin{align*}
0000000000001010 &\quad \leftarrow 21 \\
+ 111111111101010 &\quad \leftarrow -14 \\
\hline
1000000000000111 &\quad \leftarrow 7(16 ビット) 11111111111001
\end{align*}
\]

図 5: 補数を使った計算

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

5
- 正の数は 16 ビット 2 进数でそのままの表现です。一方、负の数は 2 の补数を使います。正か负がる判断は、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 であれば负です。
- 最上位のビットが符号を表すため、絶対価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。

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

- COMET II の符号无し整数

- 正の数は 16 ビット 2 进数でそのままの表现です。一方、负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 进数と同じです。したがって、表現可能な整数は 0〜65535 です。

\[
\begin{align*}
\text{最小値} & \quad (0000000000000000)_2 = (0)_{10} \\
\text{最大値} & \quad (1111111111111111)_2 = (2^{16} - 1)_{10} = (65535)_{10}
\end{align*}
\]

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)。文字とそれぞ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。

<table>
<thead>
<tr>
<th>address</th>
<th>data</th>
<th>16進数</th>
<th>文字</th>
</tr>
</thead>
<tbody>
<tr>
<td>A F F F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 0 0 0</td>
<td>0 0 0 0 0 0 0 1 0 1 1 0 0 1</td>
<td>0 0 5 9</td>
<td>Y</td>
</tr>
<tr>
<td>B 0 0 1</td>
<td>0 0 0 0 0 0 0 1 1 0 0 0 0 1</td>
<td>0 0 6 1</td>
<td>a</td>
</tr>
<tr>
<td>B 0 0 2</td>
<td>0 0 0 0 0 0 0 1 1 0 1 1 0 1</td>
<td>0 0 6 D</td>
<td>m</td>
</tr>
<tr>
<td>B 0 0 3</td>
<td>0 0 0 0 0 0 0 1 1 0 0 0 0 1</td>
<td>0 0 6 1</td>
<td>a</td>
</tr>
<tr>
<td>B 0 0 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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）。

<table>
<thead>
<tr>
<th>メモリー</th>
<th>16進数</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 0 1</td>
<td>← 0 0 0 9 ← (9)_{10}</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1</td>
<td>← 0 0 3 9 ← &quot;9&quot;</td>
</tr>
</tbody>
</table>

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ッシュのようなものです。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

<table>
<thead>
<tr>
<th>記号</th>
<th>語源</th>
<th>日本語</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR</td>
<td>General Register</td>
<td>汎用レジスタ</td>
<td>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。</td>
</tr>
<tr>
<td>SP</td>
<td>Stack Pointer</td>
<td>スタックポインタ</td>
<td>スタック領域の最上段のアドレスを保持する。</td>
</tr>
<tr>
<td>PR</td>
<td>Program Register</td>
<td>プログラムレジスタ</td>
<td>次に実行する命令のアドレスを保持する</td>
</tr>
<tr>
<td>FR</td>
<td>Flag Register</td>
<td>フラグレジスタ</td>
<td>演算結果の状態を保持する</td>
</tr>
</tbody>
</table>

- 汎用レジスタ
  * 計算等に主に用いられる。
  * 16ビットのレジスタが8個（GR0〜GR7）ある。

- スタックポインタ
  * 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て，次のようになる。

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

- 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、汎用レジスタを示す。ここに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 GR1 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

<table>
<thead>
<tr>
<th>PGM</th>
<th>START</th>
<th>;プログラムの開始</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>GR1 , A</td>
<td>;アドレス A の値を汎用レジスター G R 1 に格納する</td>
</tr>
<tr>
<td>ADDA</td>
<td>GR1, B</td>
<td>;アドレス B の値と G R 1 の値を累加</td>
</tr>
<tr>
<td>ST</td>
<td>GR1, C</td>
<td>;G R 1 に格納されている値をアドレス C に入れる</td>
</tr>
<tr>
<td>RET</td>
<td>;呼び出し元へ戻る</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>DC 3</td>
<td>;メモリを確保して、3 を格納する</td>
</tr>
<tr>
<td>B</td>
<td>DC 5</td>
<td>;メモリを確保して、5 を格納する</td>
</tr>
<tr>
<td>C</td>
<td>DS 1</td>
<td>;メモリを 1 ワード 分確保する</td>
</tr>
<tr>
<td>END</td>
<td>;プログラムの終了</td>
<td></td>
</tr>
</tbody>
</table>

主記憶装置

<table>
<thead>
<tr>
<th>adress</th>
<th>data(2進数)</th>
<th>data(16進数)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0110010010101000</td>
<td>101</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>1101101010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0010001010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1010001010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0001001010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1001001010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0010001010100000</td>
<td>100</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1010001010100000</td>
<td>100</td>
</tr>
<tr>
<td>A 0 0 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 0 0 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 0 1 0</td>
<td>0001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 0 1 1</td>
<td>1001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 1 0 0</td>
<td>0010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 1 0 1</td>
<td>1010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 1 1 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>A 1 1 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 0 0 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 0 0 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 0 1 0</td>
<td>0001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 0 1 1</td>
<td>1001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 1 0 0</td>
<td>0010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 1 0 1</td>
<td>1010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 1 1 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>B 1 1 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 0 0 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 0 0 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 0 1 0</td>
<td>0001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 0 1 1</td>
<td>1001001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 1 0 0</td>
<td>0010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 1 0 1</td>
<td>1010001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 1 1 0</td>
<td>0000001010100000</td>
<td>000</td>
</tr>
<tr>
<td>C 1 1 1</td>
<td>1000001010100000</td>
<td>000</td>
</tr>
</tbody>
</table>

図 9: アセンブリ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言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。

<table>
<thead>
<tr>
<th>ラベル欄</th>
<th>命令コード欄</th>
<th>オペランド欄</th>
<th>注釈欄</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGM</td>
<td>START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>GR1,A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDA</td>
<td>GR1,B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>GR1,C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>D,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>DC</td>
<td>3</td>
<td>;アドレスAに3を格納</td>
</tr>
<tr>
<td>B</td>
<td>DC</td>
<td>5</td>
<td>;アドレスBに5を格納</td>
</tr>
<tr>
<td>C</td>
<td>DS</td>
<td>1</td>
<td>;アドレスCから1語分の領域確保</td>
</tr>
<tr>
<td>D</td>
<td>DC 'END'</td>
<td></td>
<td>;アドレスDから文字&quot;END&quot;を格納</td>
</tr>
<tr>
<td>E</td>
<td>DC 3</td>
<td></td>
<td>;アドレスEに3を格納</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

図10：CASL IIのプログラムの書き方