教育用アセンブラCASLの開発

平山　弘

Development of the Assembly Language CASL for Education

Hiroshi HIRAYAMA

Abstract

A cross-assembly language CASL of COMET computer is implemented on the personal computer NEC PC-9801 for the education of an assembly language. Related utilities, cross linker and simulater of COMET are also implemented. Coding of these program in FORTRAN 77 makes easy to run them on various computers. And object files produced by the cross assembler and a load module which is output of cross linker are character files for understanding of object and load module. It is found that this CASL assembler system is very good for education of assembly language for a beginner though it has defects.

1. まえがき

コンピュータ教育の中でのアセンブリ言語の必要性は，いろいろな所で論じられている。アセブリ言語を修得するためには，コンピューターのアーキテクチャー，メモリ，レジスタ等のハードウエアに関する知識，2進・10進・16進数についての知識，AND・OR・SHIFTなどの論理演算に関する知識などが要求される。これらの内容はたとえ高級言語だけを使う場合でもコンピューターを使いこなすためには必ず理解していなければならない事柄である。コンピューターによって計測機器等を制御するためにも，これらの内容を理解している必要がある。このため，通産省の行なっている情報処理技術者の試験でも，アセンブリ言語が一部必修になっている。

アセンブリ言語の教育では，アセンブリ言語がコンピューター毎にその内容が異なるため，どのコンピューターを選ぶかが大きな問題となっている。大型計算機を使うことも考えられるが，大型計算機は，非常に複雑なオペレーティングシステムを持っているため，アセンブリ言語でプログラムを書くためにはある程度それを理解する必要がある。このため，初心者を対象とする教育には適さないように思える。

また，実際の応用で使われているマイクロコンピューターを使うことも考えられる。これらのものとしては，インテル社の8080・8086，モトローラ社の68000，ザイログ社のZ80等が考えられる。いずれのマイクロコンピューターでも完全に使いこなすには，かなりの時間が必要である。時間が十分にあるならば，これらのマイクロコンピューターを使うのも一方法と思われる。

さらに，仮想コンピューターを使うことも考えられる。これらのものとして考えられるのが，通産省の情報処理技術者の試験で使われている仮想コンピューターCOMETのアセンブリ言語のCASLである。このコンピューターは非常に単純な構造を持ち，命令も30命令と少なく，疑似命令も大変少ない。初心者にも容易に修得しやすいように思える。それ以前に使われていた仮想コンピューターCOMP-Xと比較して，コンピューターのアーキテクチャーや命令が現実のコンピューター近くになり，大変使い易くなっている。

このようなことから，仮想コンピューターCOMETを教育用として使ってみるため，COMET用のアセンブラ，リンカーおよびシミュレータを作成した。そして，その評価を行なった。

アセンブリ言語CASLおよび仮想コンピューターCOMETの仕様は，昨年（昭和61年）に発表され，今年（昭和62年）の情報処理技術者試験から使われ始めている。このため，昨年から今年にかけて多数のコンピューター関連雑誌で，CASLおよびCOMETのアセ

昭和62年9月28日受理

ンブラ，シミュレーションが発表されている。これらの中でソースプログラムとして発表されているのは，いずれもパーソナルコンピューターのBASIC言語で書かれたもので，かなりそのパーソナルコンピューターに依存するようなプログラムである。このため，これを大型コンピューター等で使うことは殆ど不可能に近い。大型コンピューターで使うことを考慮して，今回作成したアセンブラおよびシミュレータは，標準的なFORTRAN 77で，ほぼすべて記述した。

2. アセンブリ言語CASLシステムの構成

これらのプログラムを作成するのに，特に留意した点は，以下の3点である。第一は，教育用として開発したものであるが，あまり極端簡略化を行なわないことである。なるべく通常のアセンブリ言語を使うような形式で使えるようにすることである。これらのプログラムも，オペレーティング・システムが準備しているコマンドのように使うことができるようにした。すでに存在するプログラムと共存させるためでもある。

このようにすれば，アセンブリ言語がコンピューターシステムの中でどの様な位置にあり，どの様な役割果たしているのかを理解しやすいと考えたからである。コンピューター・システムを理解することも重要だと思われるからである。アセンブリ言語だけを使うのであるならば，エディター，リンカー等をその中に含ませるほうが使いやすい。しかし，コンピューター・システムに直接触れるようにしたほうが，コンピューターの学習者にとってより有益だと思われたので，アセンブラ，リンカー，シミュレーターを一つ一つ作成した。したがって，アセンブリ言語CASLシステムは

(1) CASLアセンブラ
(2) CASL用リンカー
(3) COMET用シミュレータ

の3つのプログラムで構成した。

第二は，通常のアセンブリ言語では見ることが出来ないような部分もなるべく見ることが出来るようにすることである。アセンブルの意味，オブジェクトコードの構造等を理解し易いように，オブジェクト・コードは文字型ファイル形式とした。ロードモジュールも簡単にブリンター等に出力させて内部を見ることが出来るように，同様に文字型ファイル形式とした。

このような形式にすれば，処理速度が落ち，ファイルの大きさが大きくなるなど非効率な部分が出てくるが，このプログラムの使用目的から，大きなプログラムを作ることはないと考えられるので，分かりやすさのほうを優先させた。

第三は，大型コンピューターや他のコンピューターに容易に移植出来るように作成した点である。このため，プログラムはごく一部を除いて，すべてFORTRAN 77の範囲で記述し作成した。現時点では，パーソナルコンピューターおよび大型コンピューターともに使える言語は，FORTRANしか考えられなかったからである。

これらのプログラムの中でFORTRAN 77から外れる部分は，次の3点である。

(1) コマンドラインを得るために，システムサブルーチンを使用している。

(2) 時間と日付を得るために，システムサブルーチンを使用している。

(3) タブコードとして，16進数の9を使っている。ASCIIコードを使っていないコンピューターでは変更が必要である。

これは，FORTAN 77の規格外のことであるが，整数形変数は32ビット以上の精度があるとしてプログラムを作成している。

このような，システム関係のプログラムではC言語が向いていると言われているが，このプログラム作成開始時点で，当大学の計算センターでは，C言語が使えなかったので，FORTRANを選んだ。

3. プログラムの外部仕様

プログラムの起動は，オペレーティングシステムのコマンドを実行するような形式とした。アセンブラを例にとり，プログラムの実行開始方法を説明する。

アセンブラは，コマンドの入力行からアセンブルの指示を与える。オブジェクト・ファイル名やリスト・ファイル名もここで指示する。この場合，オブジェクト・ファイル名やリスト・ファイル名は省略出来る様になっているものが多い。ここで作成したアセンブリ言語CASLでも，この様な点は省略出来るように作成した。その方法は，DEC社のコンピューターVAXのオペレーティングシステムVMSを参考にした。アセンブルの指示は，MS-DOSを使っている場合，次のような形式で行なう。

A>CASL SOURCE (3・1)

この場合，“SOURCE” という名前のファイルをアセンブルせよという意味になる。実際にアセンブルされるファイル名は，MS-DOS をはじめ，UNIX 等のオペレティング・システムで行なわれるように，“SOURCE” の名前に拡張子が付加された名前が使われる。オブジェクト・ファイルおよびリスト・ファイル名の指定も，この名前に違った拡張子を付加した名前を持つファイルが自動的に指定される。このために，CASL では，拡張子が省略された場合，次のような拡張子を使うように作られている。

ソース・ファイル …… CAS
オブジェクト・ファイル …… COB
リスト・ファイル …… LIS

基本的な使い方をする場合は，（3•1）のコマンドの形だけで十分であるが，高度な使い方をするために，スイッチと呼ばれるオプション機能がある。これは次のような形式を持っているものである。

/〈キーの名前〉[＝〈キーの値〉]　（3・2）

〈キーの名前〉は機能を変更するために使われる名前である。〈キーの値〉はその変更のために値が必要な場合に指定する部分である。[ ]で囲まれた部分は省略可能な部分を意味する。キー値がないような例としては次のようにものがある。オブジェクト・コードを出力しないことを指定する場合，

/NOOBJECT　（3・3）

のように指定する。キーの値が必要な例として，リスト・ファイルを “A. LST” に変更する事を指定する場合，

/LIST＝A. LST　（3・4）

と指定する。（3•3）および（3•4）の機能変更を行う場合，（3・1）は次のように指定される。

A>CASL/NOOBJECT/LIST＝A. LST SOURCE　（3・5）

さらに，キーの指定の省略形を許すようにした。その省略方法は，他のキーと区別できる範囲で上から何文字かをとるという方法である。この省略方法を採用したのは，通常の省略方法である本来の名前の外に，省略形の名前を別に用意されている場合に比べ，非常に単純で覚え易いと思われたからである。省略形を別に用意されている場合，その省略形を作る方法はある程度規則はあるとしても，完全でないため，覚えにくいためせっかく準備された省略形が利用されず，操作性が良くならないように思えたからである。CASL の LIST キーの場合，次の 4 通りの指定が可能である。

/L＝A. LST　（3・6）
/LI＝A. LST　（3・7）
/LIS＝A. LST　（3・8）
/LIST＝A. LST　（3・9）

NOOBJ のキーの場合，NOLIST という似た名前のキーがあるため，最低でも 3 文字指定しなければならない。したがって，

/NOO　（3・10）
/NOOB　（3・11）
/NOOBJ　（3・12）
/NOOBJE　（3・13）
/NOOBJEC　（3・14）
/NOOBJECT　（3・15）

の 6 通りの指定が可能となる。

キーの値で指定されるのは，ファイル名が多い。このファイル名の指定にも，省略形を許すようにした。拡張子を省略した場合，自動的に付加するようにした。付加する拡張子は，ファイル名が省略された時に付加されるものと同じである。例として，オブジェクト•ファイルの出力先を指定してみると

/O＝A　（3・16）

となる。キー “O” は，OBJECT の省略形である。この場合，オブジェクト•ファイルの省略拡張子は “COB” だから，出力先を “A. COB” に指定したことになる。

リンカーおよびシミュレータである CLINK および CSIM についても，まったく同様な使い方が出来るようにした。シミュレータについては，その中で使われるサブコマンドについても，この様な省略形を許すようにした。

4. アセンブリ言語 CASL

アセンブリ言語 CASL の仕様[1] では，そのオブジェクト・コードについては何も規定していない。参考資料としてその例が示されているだけである。したがって，そのオブジェクト・コードは自由に設定出来る訳だが，参考資料に載っているコードをそのまま使うことが，他の CASL アセンブラでも行なわれているので，ここでもそのまま使うことにした。

マクロ命令に対しては，参考資料の中である程度例示されているが，CASL アセンブリ言語毎にその実現方法は異なっている。ここで作成した CASL では，マクロ命令も普通の命令と同様な命令として扱うことにした。IN および OUT 命令は 4 語使う命令とし，第 1 語には命令コードを入れ，第 2 語には第 1 オペランド値を入れる。第 3 語は使用しないで，第 4 語に第 2 オペランドの値を入れるようにした。IN および OUT 命令の第 3 語は，命令はすべて 2 語であるという CASL の仕様から 2 語の倍数に調整するために，ダミーに付加したものである。EXIT 命令は 2 語の長さを持つ通常の命令のように設定した。EXIT はオペランドがないので，第 2 語には何も入れないことにした。この様に設定することは，ハードウエア的には，存在しない命令を実行しようとして，割り込みが起こり，その割り込みルーチンで，IN, OUT, EXIT のような機能が実行されると考えられる。

末定義ラベルについては，一部拡張してある。CASL の仕様では，DC の疑似命令の次にだけ，末定義ラベルを置くことができるが，このシステムでは，アドレスが決定出来ないラベルはすべて末定義ラベルとしている。このため，CALL 命令の後に続く名前を末定義ラベルとして扱えるので，FORTRAN のような使い方が出来るので大変便利である。細かいことであるが，ソースプログラムにタブコードを使えるようにも拡張してある。

エラーに対しては，そのエラー内容を表示すると同時に，そのエラーが発生した行番号も表示するようにした。また，エラーが発生してもその行でアセンブルをやめずに最後まで行なうようにした。エラー対策については，このプログラムが完成して，まだ時間があまりたっていないことから，十分なテストが済んでい

```
***  CASL LISTING (   IKUTOKU TECHNICAL UNIVERSITY   ) ***    PAGE: 01
                                                    DATE  : 87-09-24 14:33:31
                                                    SOURCE: FNUMB.CAS

 NO. ADDR    CODE            SOURCE
0001                 ; フィボナッチ数を求めるプログラム
0002                 ; 入力：ＧＲ１  何番目のフィボナッチ数を求めたいかを
0003                 ;                ＧＲ１出与える。
0004                 ; 出力：ＧＲ２  フィボナッチ数列の第ｎ番目項の値
0005 0000            FNUMB   START
0006 0000 4010001E           CPA     GR1,CONST3
0007 0002 60000008           JPZ     NEXT
0008 0004 1221FFFF           LEA     GR2,-1,GR1
0009 0006 81000000           RET
0010 0008 70010000   NEXT    PUSH    0,GR1
0011 000A 1211FFFF           LEA     GR1,-1,GR1
0012 000C 80000000           CALL    FNUMB
0013 000E 70020000           PUSH    0,GR2
0014 0010 1211FFFF           LEA     GR1,-1,GR1
0015 0012 80000000           CALL    FNUMB
0016 0014 1120001F           ST      GR2,WRK
0017 0016 71200000           POP     GR2
0018 0018 2020001F           ADD     GR2,WRK
0019 001A 71100000           POP     GR1
0020 001C 81000000           RET
0021 001E 0003       CONST3  DC      3
0022 001F 0000       WRK     DS      1
0023 0020                    END

   *** LABEL ***

 NO.   NAME      ADDR (HEX)        ADDR (DEC)
0001   FNUMB     0000              00000
0002   NEXT      0008              00008
0003   CONST3    001E              00030
0004   WRK       001F              00031
0005   $$$END    0020              00032
```

Fig. 1. The listing produced by the CASL assembler.

るとは言えない。まだ，多くの不十分なところがあるのではないかと考えている。これらは，徐々に解決しなければならない問題である。

アセンブルの例として，CASL の仕様書にもある CASL プログラムのアセンブル・リストを図 1 に示した。

5. オブジェクト・モジュールの構造

アセンブリ言語の言語仕様には，オブジェクト・モジュールについては，なにも規定していない。ここで作成したアセンブリ言語 CASL では，つぎのような構造にした。

(1) プログラムの入口名（外のサブルーチンから呼ばれるときの名前）
(2) プログラムの実行開始ラベル名
(3) プログラムの実行開始アドレス（このモジュールの先頭からの相対アドレスで 16 進表示）
(4) オブジェクト・モジュールの長さ（語単位で 10 進表示）
(5) 外部参照名の数（10 進表示）
(6) 外部参照名（外部参照名の数だけ繰り返される。）
(7) オブジェクト・コードとそれを定義しているソースプログラムの行番号（オブジェクト・コードは 16 進，行番号は 10 進表示）

オブジェクト・コードの最初の数字は，つぎのような意味を持っている。

1……この命令は 2 語の長さを持ち，リンクによって変化しない。
2……この命令は 2 語の長さを持ち，アドレス部はこのモジュールの先頭からの相対アドレスとなっている。
3……この命令は 2 語の長さを持ち，アドレス部は外部名を参照している。アドレス部には，外部名のオブジェクト・モジュールの(6)に於ける位置となっている。
4……この命令は 1 語の長さを持ち，リンクによって変化しない。
5……この命令は 1 語の長さを持ち，アドレス部はこのモジュールの先頭からの相対アドレスとなっている。
6……この命令は 1 語の長さを持ち，アドレス部は外部名を参照している。アドレス部には，外部名のオブジェクト・モジュールの(6)に於ける位置となっている。
7……文字列であることを示す。次の 4 文字で文字列の長さ（16 進表示）を表わし，さらに次の 4 文字で文字保存領域のアドレスを示す。さらに，次の行からその文字列が続く。
8……領域確保することを意味する。次の 4 文字でその領域の長さを 16 進数で示す。
9……IN, OUT 命令専用で，16 進数の 12 桁で命令，第一オペランド，第二オペランドを示す。

```
FNUMB
FNUMB
0000
    32
     0
24010001E          6
260000008          7
11221FFFF          8
181000000          9
170010000         10
11211FFFF         11
280000000         12
170020000         13
11211FFFF         14
280000000         15
21120001F         16
171200000         17
22020001F         18
171100000         19
181000000         20
40003             21
80001             22
```

Fig. 2. The object module produced by the CASL assembler.

アセンブルリストを示した同じ例について，オブジェクト・コードを図 2 に示した。

6. リンカーCLINK および実行モジュール

アセンブリ言語 CASL ではリンカーについても，何も規定していない。CASL ではメインプログラムという概念がない。すべてサブルーチンである。このため，リンクを行なうためには，メインプログラムを指定しなければならない。CLINK では，この指定をリンカーのオペランドで指定することにした。最初のオペランドで指定したモジュールがメインルーチンとなる。

CLINK では，アセンブラと同様にファイルの拡張子を省略出来るようになっている。CLINK で使われている拡張子は，つぎの 3 つである。

オブジェクト・ファイル　…… COB
実行モジュール・ファイル　…… CEX

```
      0
     32
4010
001F
6000
0008
1221
FFFF
8100
0000
7001
0000
1211
FFFF
8000
0000
7002
0000
1211
FFFF
8000
0000
1120
001F
7120
0000
2020
001F
7110
0000
8100
0000
0003
0000
```

Fig. 3. The load module produced by the CLINK linker.

マップ・ファイル …… CMA

これを利用して，次のような形で指定できる。

A>CLINK A B C

これは，A. COB, B. COB および C. COB をリンクして，実行モジュールを作成し，A. CEX に出力することを意味する。

CLINK では，実行モジュールを出力するだけでなく，リンクの状況を示すリンクマップも出力する。

実行モジュール・ファイルは，次の構造にした。

(1) 実行開始アドレス（10 進表示）
(2) 機械語の長さ（8 ビット単位，10 進表示）
(3) 機械語（8 ビット単位，16 進 4 桁表示）

図 2 の例についての実行モジュールを図 3 に示す。

7. シミュレータ CSIM

仮想コンピューターCOMET のシミュレータである。これは，アセンブリ言語でプログラムを開発するときに使われる，デバッガーとほぼ同じ内容を持つものである。このプログラムでは，作成したプログラムを実行するだけでなく，そのプログラムの細かい部分も検証が出来るように作られている。このシミュレータは，11 個のサブコマンドをもっている。以下にそれを示す。このサブコマンドも CASL のキーと同様な省略形を持っている。下線部分が最小限指定しなければならない部分である。

(1) REGISTER
レジスター，プログラムカウンター，フラグおよび次の命令を内容を表示する。

(2) UNASSEMBLE
指定した開始アドレスから指定したステップ数だけ逆アセンブルする。

(3) DUMP
指定した開始アドレスから，指定したステップ数だけ 16 進ダンプする。

(4) ASSIGN
指定したアドレスからメモリの内容を変更する。

(5) LOAD
指定したファイルからプログラムをメモリにロードする。

(6) GO
現在のプログラムカウンターからプログラムの実行を開始する。

(7) TRACE
指定したステップ数だけ，レジタスの内容を表示しながらプログラムを実行する。

(8) BREAK
ブレークポイントの設定，解除および表示を行なう。

(9) LOG
ログファイルのオープン・クローズを行なう。

(10) STACK
スタックの内容を表示する。

(11) QUIT
シミュレターを終らせる。

さらに，レジタス名，メモリを指定することによってその値を表示させることが出来る。代入文を書くことによってそのメモリ，レジタスの内容を変えることができる。メモリの指定は MEMORY という配列が在るかのように指定すればよいようになっている。MEMORY の指定にも省略形が許されているので，それを使えば，100 番地の内容を 16 進数で 1A4 に変更するには

```
COMET>LOAD FNUMB
COMET>REG
 PC=0000 GR0=0000 GR1=0000 GR2=0000 GR3=0000 GR4(SP)=0FA1 FR=0
 CPA   GR1,001E
COMET>GR1=7
 GR1 = 0007(HEX)       00007(DEC)
COMET>G
 *** STACK OVER ***
 PC=001E GR0=0000 GR1=0007 GR2=0008 GR3=0000 GR4(SP)=0FA1 FR=0
?
COMET>GR2
 GR2 = 0008(HEX)       00008(DEC)
COMET>QUIT
```

Fig. 4. The log file produced by the CSIM simulator of the COMET computer.

COMET＞M(100)＝#1A4

と指定すれば良い。"#" は 16 進数を示す記号である。シミュレータ実行のようすを図 4 に示した。

8. アセンブリ言語 CASL について

CASL は先にも述べたように情報処理技術者試験のためのアセンブラである。通常のコンピュータと比較してかなり単純になっている。このため，COMET コンピューターでは，レジスター間の演算は殆ど出来ない。今までアセンブラを利用してきた者にとって，非常に書きにくく思えた。アセンブラでプログラムを書く大きな目的の 1 つは，高速実行だからレジスタをなるべく使うように訓練されてきたからである。また，コンピューターもその様に設計されているからである。

アセンブラ教育の目的の一つに，割り込みの学習がある。COMET には割り込み関係の命令およびそれを実行するための構造がないため，このような目的に COMET を使うにはそれらの命令を付加させる必要がある。

このような欠点はあるにしても，単純な構造を持っているため，そのアーキテクチャ等を理解することは容易で，アセンブラの最初の学習には大変良いように思える。

参 考 文 献

1) 情報処理セミナー編集部：情報処理試験のアセブラ言語 CASL 新紀元社（1987）
2) 小島　進：MC 68000 の使い方　オーム社（1982）
3) Stephen P. Morse（内藤　祥雄訳）：8080 入門　システムソフト（1981）