T-11用クロスアセンブラast11使用説明書

T-11クロスアセンブラ（ast11）はPDP-11（μT-11）のオブジェクトコードを生成するアブソリュートアセンブラである。VAX-11、SUN-3、SUN-4、PC9801の各マシン上で動作し、全く同一のフォーマットのオブジェクトファイルを生成する。

目 次

1 コマンドとしての使い方 **2**

2 言語仕様 **2**

2.1 使用する文字セット 2
2.2 式と名前 2
2.3 プログラムの構造 4
2.4 ラベル定義部 4
2.5 定数定義 5
2.6 命令または疑似命令 5
2.6.1 命令ニーモニック 6
2.6.2 オペランド 6
2.7 疑似命令 7
2.8 マクロ命令 9
2.9 注釈 9

3 エラーメッセージ **9**

3.1 致命的エラー 9
3.2 通常のエラー 9
3.3 内部エラー 10

1　コマンドとしての使い方

文法： **ast11** [options] source[.t11]

ただし、*source* はソースファイル名。拡張子を省略した場合で、ファイル *source* が存在しない時には拡張子 '.t11' を自動的に補う。

オプション：

-o *object*	オブジェクトファイルの名前を *object* とする。（デフォルトは *source*.o11）
-g	オブジェクトファイルにシンボル情報を付け加える。
-l *list*	リスティングファイル *list* を生成する。エラーが発生した場合は（エラーメッセージを含んだ）リスティングファイルの名前は強制的に *source*.err となる。
-s	リスティングファイルの最後にシンボル表を付け加える。（**-l** オプションを指定したときのみ許される。）

2　言語仕様

ソースプログラムの記述法について説明する。これより以後、文法の説明にはバッカス記法（ＢＮＦ）を用いる。終端記号は"（引用符）でくくって示す。終端記号のうち複数の英文字より成るもの（予約語）については大文字と小文字の区別はしない。

2.1　使用する文字セット

ソースプログラムの記述には ASCII 印字文字および空白 (ASCII コード $040_{(8)}$)、タブ (ASCII コード $011_{(8)}$) および改行コード（システムによって変わる。Unix[1]では ASCII コード $012_{(8)}$、MS-DOS[2]では ASCII コード $015_{(8)}$ と ASCII コード $012_{(8)}$ の２文字）を用いる。

2.2　式と名前

プログラム中で用いる式の文法は次の通りである。

```
式　::=　一次子｜式　演算子　一次子

一次子　::=　数値定数｜定数名｜"."｜"-"　一次子｜"<"　式　">"

数値定数　::=　8 進定数｜10 進定数｜1 文字定数｜2 文字定数

8 進定数　::=　8 進数字｜8 進定数　8 進数字

8 進数字　::=　"0"｜"1"｜"2"｜"3"｜"4"｜"5"｜"6"｜"7"

10 進定数　::=　10 進数字｜10 進定数　10 進数字
```

[1] Unix は米国 AT&T ベル研究所の米国における商標である。
[2] MS-DOS は米国マイクロソフト社の登録商標である。

10 進数字 ::= 8 進数字 | "8" | "9"

1 文字定数 ::= "'"文字

2 文字定数 ::= " "文字 文字

定数名 ::= 名前

名前 ::= シンボル名 | ローカルシンボル名

シンボル名 ::= 英字 | 名前 英数字

ローカルシンボル名 ::= 10 進定数"$"

英字 ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i"
| "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r"
| "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
| "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I"
| "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R"
| "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

英数字 ::= 英字 | 10 進数字

演算子 ::= "+" | "-" | "*" | "/" | "!" | "&"

- 一次子としての"."の値は行の先頭におけるロケーションカウンタの値となる。
- 8 **進定数**および 10 **進定数**の値の範囲は 0 から $177777_{(8)}$（すなわち $65535_{(10)}$）までである。
- 1 **文字定数**および 2 **文字定数**の定義中で現われている非終端記号**文字**は任意の ASCII 印字文字、空白、タブのいずれかである。
- **行**の長さは 1023 文字までである。
- **シンボル名**の長さも同じく 1023 文字までであるが、先頭の 9 文字までしか識別しない。
- **シンボル名**の中の大文字、小文字は区別しない。
- **ローカルシンボル名**が有効となるのは**シンボル名**に対するラベル定義[3]によって区切られる範囲である。例えば、

```
        ...
1$:                     ;(A)
        ...
        BR      1$
```

[3] 2.4 参照

```
LABEL:
        ...
1$:                         ;(B)
```

において、BR 命令のオペランドの**ローカルシンボル名** 1$は行 (A) の番地を意味する。

- **演算子**のうち、“*” は乗算を、“/” は除算を、“!” はビットごとの OR を、“&” はビットごとの AND をそれぞれ表わす。
- **演算子**の間には優先順位はなく、すべて左から右に処理する。優先順位を変えるには “<” と “>” によるかっこを用いる。
- 演算の結果は 16 ビットでまるめられる。したがって例えば、“−65535” は “1” と同じになる。

2.3 プログラムの構造

プログラムは行の集りである。

プログラム ::= 空 | プログラム 行

行と**行**の間は改行コードによって区切る。
また、**行**は次の 3 つの部分より成る。

行 ::= ラベル定義部 命令または疑似命令 注釈

それぞれの部分は省略可能である。(空行も許される。)
以下、それぞれの部分について説明する。

2.4 ラベル定義部

ラベルの定義は、

ラベル定義部 ::= 空 | ラベル名 ":" ラベル定義部

ラベル名 ::= 名前

の形で書く。**ラベル名**に対する値としてこの行の番地 (ロケーションカウンタの値) を定義する。

- 1 行に複数のラベルを定義してよい。
- いったん定義した値を変更することはできない。
- 予約語を**ラベル名**として用いることはできない。

2.5 定数定義

定数定義 ::= 定数名定義｜レジスタ定数定義｜ロケーションカウンタ定義

定数名定義 ::= 名前 "=" 式

レジスタ定数定義 ::= レジスタ定数名 "=" レジスタ名

レジスタ定数名 ::= 名前

レジスタ名 ::= 定義ずみレジスタ名｜レジスタ定数名

定義ずみレジスタ名 ::= "%" 8 進数字
｜"R0"｜"R1"｜"R2"｜"R3"
｜"R4"｜"R5"｜"R6"｜"R7"

ロケーションカウンタ定義 ::= "." "=" 式

- **定数名定義**は**名前**に値を定義する。値を再定義することはできない。したがって、

```
LABEL:
```

は本アセンブラでは

```
LABEL   =       .
```

と全く同等である。

- **レジスタ定数定義**は**名前**をレジスタ名として定義する。例えば、

```
PTR     =       R0
```

とすると、以後のプログラム中で PTR は R0 と全く同様に使えるようになる。この場合も名前の再定義はできない。

いずれの定義も、名前を実際に使用する前に行なわなければならない。

- **ロケーションカウンタ定義**はロケーションカウンタの値を**式**の値に設定する。従って、例えば

```
.       =       2000
```

とすると、以後のプログラムは 2000 番地から置かれることになる。**式**の中に現れる**名前**には、すべてあらかじめ値を定義しておかなければならない。

2.6 命令または疑似命令

アセンブリ言語の命令ニーモニックまたはアセンブラに対する指令（疑似命令）を書く。

命令または疑似命令 ::= 空｜命令ニーモニック｜疑似命令｜マクロ命令
｜定数定義

2.6.1 命令ニーモニック

このアセンブラで用いることのできる命令ニーモニックはつぎのものである。

```
命令ニーモニック  ::=  2番地部命令  2番地部オペランド
                     |1番地部命令  1番地部オペランド
                     |無番地部命令
                     |ブランチ命令  式
                     |その他の命令

2番地部命令  ::=  "MOV"|"MOVB"|"CMP"|"CMPB"
                |"BIT"|"BITB"|"BIC"|"BICB"
                |"BIS"|"BISB"|"ADD"|"SUB"

1番地部命令  ::=  "CLR"|"CLRB"|"COM"|"COMB"|"INC"
                |"INCB"|"DEC"|"DECB"|"NEG"|"NEGB"
                |"ADC"|"ADCB"|"SBC"|"SBCB"|"TST"
                |"TSTB"|"ROR"|"RORB"|"ROL"|"ROLB"
                |"ASR"|"ASRB"|"ASL"|"ASLB"|"SWAB"
                |"JMP"|"SXT"

無番地部命令 ::=  "NOP"|"HALT"|"WAIT"|"BPT"|"RTI"
                |"RTT"|"IOT"|"RESET"

ブランチ命令  ::=  "BR"|"BNE"|"BEQ"|"BGE"|"BLT"
                 |"BGT"|"BLE"|"BPL"|"BMI"|"BHI"
                 |"BHIS"|"BLO"|"BLOS"|"BVC"
                 |"BVS"|"BCC"|"BCS"

その他の命令  ::=  "JSR"  レジスタ名 "," 1番地部オペランド
                |"RTS"  レジスタ名
```

ブランチ命令をアセンブルした結果の機械語のオフセット部には、オペランドの式の値を E とすると、

$$(E - \text{ブランチ命令の次の番地})/2$$

を8ビットの符号つき数とみなした値が入る。したがって例えば、

```
br      .
```

をアセンブルした結果は、この br 命令自身に飛ぶ機械語になる。

2.6.2 オペランド

オペランドの文法は以下の通りである。

```
2番地部オペランド  ::=  1番地部オペランド "," 1番地部オペランド
```

表 1: 1 番地部オペランドのモード

モード	モード名	文法
モード 0	レジスタ直接	R
モード 1	レジスタ間接	@R または (R)
モード 2	自動増加	(R)+
モード 3	自動増加間接	@(R)+
モード 4	自動減少	−(R)
モード 5	自動減少間接	@−(R)
モード 6	インデクス	E(R)
モード 7	インデクス間接	@(R)
モード 2、レジスタ 7	内包	#E
モード 3、レジスタ 7	絶対	@#E
モード 6、レジスタ 7	相対	E
モード 7、レジスタ 7	相対間接	@E

（R はレジスタ名、E は式を表わす。）

```
1 番地部オペランド  ::=  基本オペランド | "@" 基本オペランド
                       | "(" レジスタ名 ")"

基本オペランド  ::=  レジスタ名
                   | "(" レジスタ名 ")" "+"
                   | "-" "(" レジスタ名 ")"
                   | 式 "(" レジスタ名 ")"
                   | 式
                   | "#" 式
```

1 **番地部オペランド** の文法とモードの対応は表 1 のようになる。

2.7 疑似命令

現在使える疑似命令は以下の通りである。

```
疑似命令  ::=  ".WORD"  式の並び | ".WORD" | 式の並び
             | ".BYTE"  式の並び | ".BYTE"
             | ".ASCII"  文字列 | ".ASCIZ"  文字列
             | ".EVEN" | ".ODD"
             | ".BLKW"  式 | ".BLKB"  式
             | ".END"  式 | ".END"
             | ".PAGE"

式の並び  ::=  式 | 式の並び "," 式

文字列  ::=  区切り文字  文字の並び  区切り文字
```

ただし、

- **区切り文字**は任意の ASCII 印字文字、**文字の並び**は**区切り文字**を含まない**文字**の並びである。
- 疑似命令として**式の並び**が現れた時には “.WORD **式の並び**” と同じように扱う。
- ”.WORD”（または”.BYTE”）の後に１つも式がないときは “.WORD 0”（または “.BYTE 0”）と同じように扱う。

各々の疑似命令について解説する。

.WORD 式の並び それぞれの式の値をワード定数とみなして順にこの場所に置いていく。

.BYTE 式の並び それぞれの式の値をバイト定数とみなして順にこの場所に置いていく。式の値は 8 ビットにまるめる。（上位バイトは無視する。）

.ASCII 文字列 文字の並びのそれぞれの文字の ASCII コードをバイト定数とみなしてこの場所に置いていく。したがって、

```
.ASCII  /ABC/
```

は、

```
.BYTE   101, 102, 103
```

と等しい。（文字 “A” の ASCII コードは $101_{(8)}$）

.ASCIZ 文字列 .ASCII と同様に**文字の並び**のそれぞれの文字の ASCII コードをバイト定数とみなしてこの場所に置いていき、最後に 1 バイトの 0 を置く。したがって、

```
.ASCIZ  /ABC/
```

は、

```
.BYTE    101, 102, 103, 0
```

と等しい。

.EVEN 現在のロケーションカウンタの値が偶数なら何の効果もない。奇数ならロケーションカウンタの値を 1 増やして偶数にする。

.ODD 現在のロケーションカウンタの値が奇数なら何の効果もない。偶数ならロケーションカウンタの値を 1 増やして奇数にする。

.BLKW 式 **式**の値のワード数だけ場所を空ける。すなわち、“.EVEN” と同じ動作を行なったのち**式**の値の 2 倍だけロケーションカウンタを増やす。**式**の中の**名前**はすべて、この時点より前に値を定義したものでなければならない。

.BLKB 式 **式**の値のバイト数だけ場所を空ける。すなわち**式**の値だけロケーションカウンタを増やす。**式**の中の**名前**はすべて、この時点より前に値を定義したものでなければならない。

.END 式 アセンブリ言語プログラムの終了を示す。**式**の値をプログラムの入口番地として宣言する。**式**を省略した時には、アセンブラはプログラムの先頭を入口番地とみなす。

.PAGE アセンブルリスティングのページを替える。この疑似命令のある行が次のページの最初の行になる。

2.8 マクロ命令

現在定義されているマクロ命令は以下の通りである。

```
マクロ命令  ::=  ".EOJ"
```

2.9 注釈

セミコロン ";" は**注釈**の始まりを示す。**注釈**は行の終りまで続く。

```
注釈  ::=  空 | ";" 文字列
```

注釈はアセンブラにとって行の終りと同じである。

3 エラーメッセージ

なんらかの原因で正常な処理ができなかった場合、アセンブラはエラーメッセージを標準エラー出力に表示する。また、**-l** オプションを指定した場合にはリスティングファイル中にもエラーメッセージを表示する。このときリスティングファイル名はユーザが指定した名前ではなくてソースファイル名の拡張子を '.err' に変更した名前となる。

エラーの種類は次の 3 種類である。

3.1 致命的エラー

ファイルをオープンできなかった場合など処理を続けられないようなエラー。この場合アセンブラは、

```
Assembler fatal error: メッセージ
```

という形式のメッセージを出力してただちに実行を終了する。このメッセージはリスティングファイルには残らない。

3.2 通常のエラー

ソースプログラムの誤りを発見した場合、

```
"ファイル名", line 行番号, メッセージ
```

という形式のメッセージを出力する。またメッセージの後にエラーを検出した文字位置を指し示す記号 'V' とともにエラーを起こしたソースファイルの行を表示する。例を挙げる。

```
"foo.t11", line 123, undefined name MOVE
          V
       move    #TAB, r0
```

このメッセージはリスティングファイルにも出力される。

3.3 内部エラー

アセンブラのプログラム自身に辻妻の合わない点があることが分かった場合、

```
>>>内部エラー番号　ASSEMBLER INTERNAL ERROR -- CALL T.A.
```

という形式のメッセージを出力し、アボート（実行を終了し、もし可能ならコアダンプ）する。このエラーが起きた場合リスティングファイルは生成されない。