Util--解説書

Ver. 1.1.0

徳山工業高等専門学校
情報電子工学科

目 次

第1章　はじめに

Util--は C--言語を TaC で実行するために必要な 5 つのツールから構成されます。

- as--は TaC 用のアセンブラです。アセンブラは C--コンパイラが出力したアセンブリ言語プログラムを、リロケータブルオブオブジェクトに変換します。

  アセンブリ言語の文法は、「付録 A as--文法のまとめ」に簡単にまとめてあります。現在のところ詳しいドキュメントがありません。「文法のまとめ」の他には、as--の動作テストに使用される Util--/AS--/test.s ファイルが参考になります。

  リロケータブルオブジェクトは、機械語プログラム、名前表、再配置情報表からなるファイルです。詳しくは、「付録 B.1 .o 形式ファイル」で説明します。

- ld--は TaC 用のリンカです。リンカは複数のリロケータブルオブジェクトを入力して、一つのリロケータブルオブジェクトに結合します。

- objexe--は、リロケータブルオブジェクトを入力し、TacOS のアプリケーションプログラムの実行形式ファイルを出力します。実行形式ファイルは、リロケータブルオブジェクトから、シンボルテーブルなどリンカしか使用しない情報を取り除いたファイルです。実行形式ファイルについては「付録 B.2 .exe 形式ファイル」で説明します。

- objbin--は TaC 用のローダです。ローダはリロケータブルオブジェクトを入力して、ロードアドレスが決定された機械語を出力します。出力ファイルの形式については「付録 B.3 .bin 形式ファイル」で説明します。ローダは、TacOS のカーネルを作成する時に使用されます。

- size--は、リロケータブルオブジェクトのテキストセグメント、初期化データセグメント、非初期化データセグメントの大きさを表示します。完成したプログラムのメモリ使用量を見積もるために使用します。

第2章　Util--のインストール

Util--は https://github.com/tctsigemura/Util--/ から入手します。インストールの準備ができたらダウンロードした配布物を解凍し Util--ディレクトリで以下のように操作します。

```
$ make
...
cc -std=c99  -DDATE="\"‘date‘\"" -DVER=\"2.1.1\" \
-o as-- syntax.c lexical.c util.c
cc -std=c99 -DDATE="\"‘date‘\"" -DVER=\"2.0.0\" -DARC=\"TaC\"
  -o ld-- ld.c
cc -std=c99 -o objbin-- objbin.c
cc -std=c99 -o size-- size.c
cc -std=c99 -o objexe-- objexe.c
$ sudo make install
...
install -d -m 755 /usr/local/bin
install -m 755 as-- /usr/local/bin
...
$
```

以上で、as--、ld--、objbin--、objexe--、size--の五つのプログラムがコンパイルされ、/usr/local/bin にインストールされました。

第3章　コマンドリファレンス

Util--に含まれる５つのツールの使用方法を解説します。

3.1　as--コマンド

as--コマンドは TaC 用のアセンブラです。TaC のアセンブリ言語で記述されたプログラムをリロケータブルオブジェクトに変換します。リロケータブルオブジェクトファイルの形式は、「付録 B.1 .o 形式ファイル」で解説してあります。

as--には手書きのプログラムを入力することもできますが、主に、C--コンパイラが出力したアセンブリ言語プログラムを入力することを想定しています。そのため、オペランドにアドレス計算式が書けない等、C--コンパイラが使用しない機能は省略されています。TaC アセンブリ言語の文法は「付録 A as--文法まとめ」の通りです。

as--コマンドの書式は次の通りです。

形式：　as-- [-h] [-v] [<ソースプログラムファイル>]

-h、-v オプションは使用法メッセージを表示します。ソースプログラムファイルの拡張子は「.s」でなければなりません。as--コマンドは拡張子が「.o」のファイルを作成し、再配置可能な機械語を出力します。次のように実行すると、“hello.o” ファイルが作成されます。

```
$ as-- hello.s
```

3.2　ld--コマンド

ld--コマンドは TaC 用のリンカープログラムです。「.o 形式ファイル」（11 ページ参照）を複数入力し、同じ形式の一つのファイルに結合します。ld--コマンドの書式は次の通りです。出力ファイルは一つだけ、入力ファイルはいくつでも指定できます。

形式：　ld-- [-h] [-v]　<出力ファイル> <入力ファイル> ...

次のように実行すると “libtac.o”、“hello.o” の二つのファイルを結合して、“mod.o” ファイルを作成します。“hello.sym” ファイルは出力されたリロケータブルオブジェクトの再配置表と名前表をダンプしたファイルです。

“mod.o” ファイルもリロケータブルオブジェクトなので、未定義シンボルを含んでいる可能性があります。

```
$ ld-- mod.o /usr/local/cmmLib/libtac.o hello.o > hello.sym
```

3.3 objexe--コマンド

objexe--コマンドは TacOS 用の実行形式ファイル作成プログラムです。「.o 形式ファイル」（11 ページ参照）を入力し、「.exe 形式ファイル」（14 ページ参照）へ変換します。objexe--コマンドの書式は次の通りです。入力ファイルも出力ファイルも一つだけ指定できます。

形式： objexe-- <exefile> <objfile> <stkSiz>

“<exefile>” は出力のファイル名です。“<objfile>” は入力のファイル名です。

“<stkSiz>” には、アプリケーションプログラムに割り付ける「スタック領域サイズ＋ヒープ領域サイズ」をバイト単位の 10 進数で指定します。

下に実行例を示します。mod.o は、ld--が出力したリロケータブルオブジェクトです。hello.map ファイルには、hello.exe 中のシンボルとアドレスの対応表がアドレス順に格納されます。シンボルのアドレスは、アプリケーションプログラムの先頭からの相対アドレスです。

```
$ objexe-- hello.exe mod.o 600 | sort --key=1 > hello.map
```

mod.o に未解決シンボルが含まれているとエラーになります。次の実行例は hello.s ファイル（hello.cmm から c--が変換したアセンブリ言語ソースプログラム）中に、未定義シンボル_printff（C--ソース中では printff）が含まれていてエラーになった例です。

```
$ objexe-- hello.exe mod.o 600 | sort --key=1 > hello.map
_printff[hello.s]:undefined symbol
```

3.4 objbin--コマンド

objbin--コマンドは TaC 用のローダプログラムです。「.o 形式ファイル」（11 ページ参照）を入力し、「.bin 形式ファイル」（15 ページ参照）へ変換します。objbin--コマンドの書式は次の通りです。入力ファイルも出力ファイルも一つだけ指定できます。

形式： objbin-- 0xTTTT <出力ファイル> <入力ファイル> [0xBBBB]

0xTTTT は 16 進数でテキストセグメントの先頭アドレスを指定します。指定した番地からテキストセグメント、データセグメントの順に配置されます。0xBBBB は 16 進数で BSS セグメントの先頭アドレスを指定します。0xBBBB は省略可能です。省略した場合、BSS セグメントはデータセグメントの直後に配置されます。

下に実行例を示します。実行例の 1 行目は “kernel.o” ファイルを入力し 0000H 番地に配置した状態の機械語を作成し “kernel.bin” ファイルに出力します。実行例の 2 行目は “ipl.o” ファイルを入力し F000H 番地にテキストセグメントとデータセグメントを連続して配置し、DA00H 番地に BSS セグメントを配置します。そして、その状態の機械語を “ipl.bin” ファイルに出力します。なお、これらの例は実物のカーネルと IPL を配置する手順です。F000H 番地からの始まる ROM 領域に IPL プログラム、RAM 領域の DA00H 番地にワークエリアを配置します。

```
$ objbin-- 0x0000 kernel.bin kernel.o | sort --key=1 > kernel.map
$ objbin-- 0xf000 ipl.bin ipl.o 0xda00 | sort --key=1 > ipl.map
```

objbin--コマンドは、固定番地に配置された完全な機械語を作成します。“kernel.o” ファイルや “ipl.o” ファイルに未定義シンボルが含まれていると完全な機械語に変換できないのでエラーが発生します。

3.5 size--コマンド

size--コマンドは「.o 形式ファイル」（11 ページ参照）または「.exe 形式ファイル」（14 ページ参照）を入力し、テキスト、データ、BSS セグメントのサイズを表示します。

形式： size-- <入力ファイル>

次のように実行します。“kernel.o” ファイルの各セグメントと全体のサイズが 10 進数と 16 進数で表示されています。

```
$ size-- kernel.o
text    data    bss     dec     filename
17436     516    4386   22338   kernel.o
(441c)  (0204)  (1122)  (5742)  (hex)
```

付 録A　as--文法まとめ

次のページに、as--の文法を BNF 風にまとめたものを掲載します。

注意１： 次ページで用いる文法表記方法
全角記号がメタ文字、意味は次の通り
(1) A ☆　は A のゼロ回以上の繰り返し
(2) 《...》はグループ
(3) 【...】は省略可能
(4) A ｜ B は A または B

注意２： 次ページを読む上での注意事項
(※0)　ラベルを宣言する時は空白なしに行頭から書く.
(※1)　ラベル行はテキストセグメントのラベルを定義する.
(※2)　EQU ラベルは定数値 (オブジェクトに出力されない).
(※3)　STRING データは TEXT セグメントの最後尾に出力される.
(※4)　ラベルに予約語と同じ綴りは使用できない.
　　’.’で始まるラベルはファイル内のローカルラベルになる.

```
プログラム：    《空行｜ラベル行｜擬似命令行｜機械語命令行》☆
空行：          【コメント】 \n
ラベル行：      ラベル 【コメント】 \n                                    (※ 1)
コメント：      ; 文字☆
擬似命令行：    EQU 命令行｜STRING 命令行｜DB 命令行｜
                DW 命令行｜BS 命令行｜WS 命令行
機械語命令行：  【ラベル】 機械語記述 \n
EQU 命令行      ラベル EQU 数値 \n                                       (※ 2)
STRING 命令行   【ラベル】 STRING 文字列 \n                              (※ 3)
DB 命令行       【ラベル】 DB 式 \n
DW 命令行       【ラベル】 DW 式 \n
BS 命令行       ラベル BS 数値 \n
WS 命令行       ラベル WS 数値 \n
機械語記述：    機械語 1｜機械語 2｜機械語 3｜機械語 4｜機械語 5｜機械語 6
機械語 1：      命令 1
機械語 2：      命令 2 レジスタ
機械語 3：      命令 3 レジスタ , 式 |
                命令 3 レジスタ , @ レジスタ |
                命令 3 レジスタ ,【 % 】 レジスタ
機械語 4：      命令 4 レジスタ , 式 |
                命令 4 レジスタ , 式 , レジスタ |
                命令 4 レジスタ , # 式 |
                命令 4 レジスタ , レジスタ |
                命令 4 レジスタ , @ レジスタ |
                命令 4 レジスタ , % レジスタ
機械語 5：      ST レジスタ , 式 |
                ST レジスタ , 式 , レジスタ |
                ST レジスタ , @ レジスタ |
                ST レジスタ , % レジスタ
機械語 6：      命令 6 式 | 命令 6 式 , インデクスレジスタ |
                命令 6 % レジスタ
レジスタ：      G0 | G1 | ... | G12 | FP | SP | USP | PC
命令 1：        NO | RET | RETI | EI | DI | SVC | HALT
命令 2：        PUSH | POP
命令 3：        IN | OUT
命令 4：        LD｜ADD｜SUB｜CMP｜AND｜OR｜XOR｜ADDS｜MUL｜
                DIV｜MOD｜MULL｜DIVL｜SHLA｜SHLL｜SHRA｜SHRL
命令 6：        JZ｜ JC｜ JM｜ JO｜ JNZ｜ JNC｜ JNM｜ JNO｜
                JNO｜ JLT｜ JLE｜ JGE｜ JGT｜ JMP｜ CALL
式：            数値 | ラベル
ラベル：        英字 英数字☆                                             (※ 4)
数値：          10 進数値 | 16 進数値 | 8 進数値 | 文字コード
10 進数値：     【-】 1..9 0..9☆
16 進数値：     【-】 0x 16 進数字 16 進数字☆
8 進数値：      【-】 0 0..7☆
文字コード：    ’ 文字 ’
文字列：        " 文字☆ "
英字：          A..Z | a..z | _ | .
英数字：        英字 | 0..9
16 進数字：     0..9 | A..F | a..f
```

付 録B　ファイルフォーマット

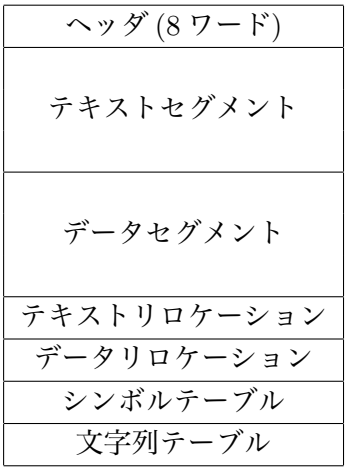
TaC 開発環境で使用する 3 種類のバイナリ形式ファイルの内容について解説します。

B.1　.o 形式ファイル

as--の出力ファイル形式です。内容は再配置可能な機械語です。ld--は、この形式の複数の入力ファイルを一つに結合します。結合されたファイルも同じ.o 形式ファイルです。

B.1.1　ファイル形式

以下に、ファイルの形式を図示します。ヘッダは各セグメントの長さ等を格納した 8 ワード固定長の情報です。テキストセグメントは機械語プログラムと文字列定数を格納します。データセグメントは初期化データを格納します。テキストリロケーションはテキストセグメントの再配置情報を、データリロケーションはデータセグメントの再配置情報を格納します。シンボルテーブルはシンボル (C--プログラムの大域変数名や関数名、アセンブリ言語のラベル) の値 (アドレス) を格納します。この情報を使用して、ld--が.o 形式ファイルの結合をします。文字列テーブルはシンボルテーブルに格納されるシンボルの綴を格納します。



.o ファイルフォーマット

B.1.2 ヘッダ

.o形式ファイルのヘッダは次の構造体により定義されます。ただし、ここで uint 型は符号無しの int 型とします。

```
struct ObjHdr {
  uint magic;     // マジックナンバー (0x0107)
  uint text;      // テキストセグメントサイズ (バイト単位)
  uint data;      // 初期化データセグメントサイズ (バイト単位)
  uint bss;       // 非初期化データセグメント (BSS) サイズ (バイト単位)
  uint syms;      // シンボルテーブルサイズ (バイト単位)
  uint entry;     // 常に 0
  uint trsize;    // テキストリロケーションサイズ (バイト単位)
  uint drsize;    // データリロケーションサイズ (バイト単位)
};
```

B.1.3 リロケーションレコード

テキストリロケーション、データリロケーション領域には、再配置情報を記録したリロケーションレコードの表が格納されます。リロケーションレコードは次の構造体で定義されます。レコードは 2 ワード長で、第 1 ワードが再配置時に書き換えが必要なポインタのセグメント中アドレス、第 2 ワードは上位 2 ビットがポインタの種類 (type) データ、下位 14 ビットがポインタ値を格納するシンボルのシンボルテーブル上の添字 (idx) データになります。

```
struct ObjRel {
  uint addr;       // 再配置すべきポインタのセグメント内アドレス
  uint type: 2,    // ポインタの型
       idx: 14;    // シンボルテーブルのポインタが登録されている位置
};

// type の意味
#define UNDEF 0    // 未定義
#define TEXT  1    // テキストセグメント
#define DATA  2    // データセグメント
#define BSS   3    // コモン
```

B.1.4 シンボルテーブル

シンボルテーブルは、シンボルとアドレスを対応付けします。アセンブラが処理したシンボル (ラベル) の中で EQU ラベルを除くものが全てシンボルテーブルに出力されます。また、アセンブラのソースプログラムのファイル名もシンボルテーブルに出力され、objexe--、objbin--プログラムが未定義シンボルを発見した場合、ファイル名とともにエラー表示ができるようにしています。ファイル名は'@' を先頭に付加してシンボルテーブルに登録されます。

シンボルは 1 文字目によって意味付けがされています。意味は、「'@' : ファイル名」、「'.' : ローカル名」です。

シンボルテーブルを構成するシンボルレコードの構成を次に示します。シンボルレコードは 2 ワード長です。第 1 ワードの上位 2 ビットがシンボルの種類 (type) を表し、下位 14 ビットが文字列テーブル上でシンボルの綴が格納されている場所を表す添字データ (sIdx) を格納します。第 2 ワード (val) はシンボルの値をセグメント内オフセットで表します。ただし、未定義 (UNDEF) シンボルの場合は 0 、コモン (BSS) シンボルの場合は領域のサイズを格納します。

ld--は、複数の入力ファイル中に同名のコモンシンボルを発見した場合、それらを一つの領域に重ね合わせます。このとき、領域のサイズは重ね合わせたシンボルの中で最大のものと同じになります。また、一つのデータセグメントシンボルと一つ以上のコモンシンボルが見つかった場合は、データセグメントシンボルに集約します。未定義シンボル同士は一つの未定義シンボルに、未定義シンボルと他の種類のシンボルは未定義ではない方のシンボルに集約します。これ以外に同名のシンボルが見つかった場合は、エラー (シンボルの 2 重定義) になります。

```
struct Symbol {
  uint type: 2,    // シンボルの型
       sIdx:14;    // シンボル名称の文字列テーブル上の位置
  uint val;        // シンボルの値
};

// type の意味
#define UNDEF 0    // 未定義
#define TEXT  1    // テキストセグメント
#define DATA  2    // データセグメント
#define BSS   3    // コモン
```

B.1.5 文字列テーブル

文字列テーブルはシンボルの綴を格納します。文字列テーブルの内容は '\0' で終端された C--言語型文字列の繰返しです。C--言語型文字列は、1 文字を 8 ビットで表現します。ヘッダに文字列表のサイズは格納されていないので、ファイルサイズから文字列テーブルのサイズを知る必要があります。

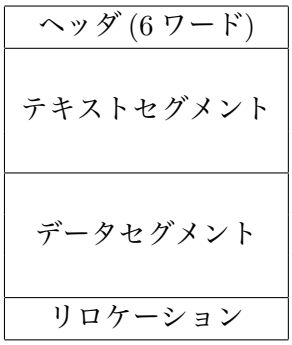
シンボルテーブルに同じ綴のシンボルが複数ある場合 (「.」で始まるローカルシンボルは同じ綴の可能性がある) は、メモリの節約のため、複数のシンボルレコードで同じ文字列表エントリーを共用します。C--コンパイラが自動的に生成するローカルラベルは、毎回、同一のパターンなので、多くのシンボルレコードで文字列表エントリーの共用がされます。

B.2 .exe 形式ファイル

TacOS の実行可能なアプリケーションプログラムファイルです。内容は、実行時に不要な情報を取り除いた再配置可能な機械語です。objexe--は、.o 形式のファイルを一つ入力して、.exe ファイルを一つ出力します。未定義シンボルを含む.o 形式ファイルは、.exe 形式ファイルに変換することができません。

B.2.1 ファイル形式

.o 形式ファイルを簡単化したファイルです。



.exe ファイルフォーマット

B.2.2 ヘッダ

.exe 形式ファイルのヘッダは次の構造体により定義されます。ただし、ここで uint 型は符号無しの int 型とします。

```
struct ExeHdr {
  uint magic;     // マジックナンバー (0x0108)
  uint text;      // テキストセグメントサイズ (バイト単位)
  uint data;      // 初期化データセグメントサイズ (バイト単位)
  uint bss;       // 非初期化データセグメント (BSS) サイズ (バイト単位)
  uint rsize;     // リロケーションサイズ (ワード単位)
};
```

B.2.3 リロケーションレコード

リロケーション領域には、再配置情報を記録したリロケーションレコードの表が格納されます。リロケーションレコードは次の構造体で定義されます。レコードは 1 ワード長で、再配置時に書き換えが必要なポインタのプログラム先頭からの相対アドレスです。

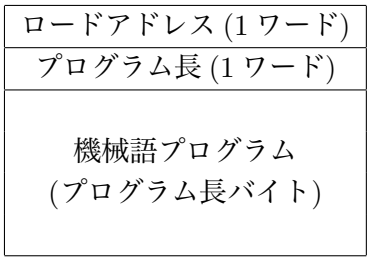
```
struct ExeRel {
  uint addr;       // 再配置すべきポインタのアドレス (バイト単位)
};
```

B.3 .bin 形式ファイル

ロードアドレスが確定した機械語プログラムを格納するためのファイル形式です。この形式のプログラムはメモリにロードするだけで実行可能です。objbin--プログラムによって.o 形式ファイルを.bin 形式ファイルに変換します。未定義シンボルを含む.o 形式ファイルは、.bin 形式ファイルに変換することができません。

B.3.1 ファイル形式

8 ビット版の TeC で使用してきた .bin 形式ファイルを単純に 16 ビットに拡張した形式のファイルです。以下に、ファイルの形式を図示します。第 1 ワードが機械語プログラムのロードアドレス、第 2 ワードが機械語プログラムの長さ (バイト単位) を表現します。第 3 ワードから先は機械語プログラム本体です。プログラム本体は、テキストセグメント、初期化データセグメント、非初期化セグメントを結合したものです。非初期化セグメントは、0x00 で満たされています。



.bin 形式ファイル

IPL プログラム (“ipl.bin”)、OS カーネルプログラム (“kernel.bin”) が、この形式のプログラムファイルです。

変更履歴

2016 年 03 月 13 日 初期バージョン

対応ソフトウェアのバージョン

AS--	Ver.2.1.1
LD--	Ver.2.0.0
OBJBIN--	Ver.2.0.0
SIZE--	Ver.2.1.0
OBJEXE--	Ver.1.0.0

Util--**解説書**

発行年月　２０１６年１０月　Ver. 1.1.0
発　　行　独立行政法人国立高等専門学校機構
徳山工業高等専門学校
情報電子工学科　重村哲至
〒 745-8585 山口県周南市学園台
sigemura@tokuyama.ac.jp