

目次

- [インラインアセンブリコードを記述する\[GCC\]](#)
- [記述したマクロが正しいか検証する\[GCC\]](#)
- [アセンブリコードを出力する\[GCC\]](#)
- [VC++用のDLLをCygwinで使う](#)
- [Qt Open Source Edition for C++/Windows 4.4.0をCygwinのGCCでリンクする\[Cygwin\]](#)
- [コンパイルができない\[GCC\]\[Cygwin\]\[Linux\]](#)
 - [エラーメッセージ中に"undefined reference to 'xxxx'"とある場合](#)

インラインアセンブリコードを記述する[GCC]

例えばx86系のCPU(IA-32アーキテクチャ)においてクロックのカウント値を得るには次のようにする。

```
unsigned int ch = 0;
unsinged int cl = 0;

__asm__ __volatile__ ("rdtsc" : "=a" (cl), "=d" (ch));
```

"rdtsc"はクロックカウント値をeaxレジスタにロードする命令 ("ReaD Time Stamp Counter") 。

"=a" (cl)は直前の命令の結果として得られたeaxレジスタの値を変数clに代入することを指示する。同様に"=d" (ch)はedxレジスタの値を変数chに代入することを支持する。この"=a"は大文字で"=A"と書くと意味が変わってしまうので注意 ("=A"だとeaxとedxを同時に読み出し、64 bit整数に代入することになる。IA-32において32bit整数を指定していた場合は暗黙のうちにedxの内容が捨てられ、eaxの内容が代入される) 。

参照:

- GCC Online Documentation (info gcc), "5.36.4 Constraints for Particular Machines"
- "Intel64 and IA-32 Architectures Software Developer 's Manual Volume 2B: Instruction Set Reference, N-Z"

記述したマクロが正しいか検証する[GCC]

次のようにして、ソースファイル (test.cとする) 中のマクロを展開した段階のソースコードを出力して確認する。

```
gcc -E test.c > test_tmp.c
```

出力はファイルtest_tmp.cに書き込まれる。

アセンブリコードを出力する[GCC]

次のように-Sオプションを指定してgccを実行する。

```
gcc -S test.c
```

出力は新たなファイルtest.sに書き込まれる。

VC++用のDLLをCygwinで使う

前提: シンボルが除去 (strip) されていないDLLもしくはそのためのインポートライブラリがあること。

VC++用のインポートライブラリをxxxx.libとすると、次の手順により、Cygwin用のインポートライブラリを作成できる。

```
echo EXPORTS > xxxx.def
nm xxxx.lib | sed -n -e"s/.* T _//p" >> xxxx.def
dlltool --kill-at --def xxxx.def --dllname xxxx.dll --output-lib libxxxx.a
```

オプション"--kill-at"が必要となるのはDLL中の関数が宣言されているヘッダファイルに、次のように__stdcallもしくはAPIENTRYが付いた関数宣言、

```
DllExport int __stdcall SomeDll_DoSomething();
```

or

```
DllExport int APIENTRY SomeDll_AnotherFunc();
```

がある場合である。

この"--kill-at"がない場合、DLLを使用するプログラムとインポートライブラリの静的リンクまでは行えるが、DLLとの動的リンクの際に失敗する（プログラムが起動中に異常終了し、エラーメッセージが出ないこともあるので注意が必要）。

"kill at"の意味通り、DLLへの参照（関数名の末尾）の"@n"が消される（nは引数の合計サイズ[byte]）。ただし、インポートライブラリ中の関数名（シンボル）の末尾には"@n"があり、これは消される対象でない（したがってnmコマンドが出力するシンボルには"@n"が付いている）。この"@n"表現はWindows/VC++の仕様上、必要なものである。

参考文献:

- [Charles Petzold, プログラミングWindows第5版, アスキー](#)（DLLの仕様はWindows98/NT4.0の頃から基本的には変わっていないので古い版にも同様の記述があるはず）

Qt Open Source Edition for C++/Windows 4.4.0をCygwinのGCCでリンクする[Cygwin]

Qtのインストール先のディレクトリをC:\Qt\として説明する。

（インストール先がこれと異なる場合は、適宜下記の説明中の該当部分を置き換えること）

1. Qtをインストールする。MinGWのインストール先についてのQtインストーラからの問い合わせ/警告は無視してよい
2. QtのツールがあるディレクトリへのパスC:\Qt\4.4.0\binを環境変数PATHに追加する

3. 次のファイル

```
C:\Qt\4.4.0\mkspecs\default\qmake.conf
```

の以下の行

```
QMAKE_MOC          = $ ${QT_INSTALL_BINS}${DIR_SEPARATOR}moc.exe
QMAKE_UIC          = $ ${QT_INSTALL_BINS}${DIR_SEPARATOR}uic.exe
QMAKE_IDC          = $ ${QT_INSTALL_BINS}${DIR_SEPARATOR}idc.exe
```

を次のように変更する。

```
MYQTDIR = /cygdrive/c/Qt/4.4.0/bin
QMAKE_CXXFLAGS += -mno-cygwin
QMAKE_LFLAGS += -mno-cygwin
QMAKE_MOC = $$MYQTDIR/moc.exe
QMAKE_UIC = $$MYQTDIR/uic.exe
QMAKE_IDC = $$MYQTDIR/idc.exe
```

4. qmakeを直接使わず、以下に示す、スクリプトファイルを用いる。

これはqmakeが出力したMakefile中のパスの形式をMinGWの形式から、Cygwinの形式に変換するためのスクリプトである。

```
#!/usr/bin/bash
echo "calling qmake..."
qmake $* > /dev/null 2>&1 || echo "Error occurred in qmake: Check your .pro file"

# unique name to avoid overwriting another file with the same name
TMPFILE="qmake_cyg_tmp_Makefile"

echo "converting Makefiles..."
for f in Makefile*;do
  cat $f |sed -e"s/\ (C \|c \|):[ \\ \| ]/\ /cygdrive \|c \|/g" > $TMPFILE; mv $TMPFILE $f;
done
rm -f $TMPFILE
```

コンパイルができない[GCC][Cygwin][Linux]

エラーメッセージ中に"undefined reference to 'xxxx'"とある場合

```
/cygdrive/c/DOCUME~1/Admin/LOCALS~1/Temp/cc668dxb.o: test.c:(.text+0x92):
undefined reference to `_xxxx'
collect2: ld returned 1 exit status
```

これを解決するには、関数xxxxが定義されているファイル、またはライブラリをGCCに対して指定する。

ライブラリ名が不明の場合、次のコマンド行により検索する（シェルがBash、かつ関数名がxxxxとする）。

```
for f in /usr/lib/*.a; do nm $f|grep "T .*xxx" && echo Found in $f; done
```

典型的な結果は次のようになる。

```
00000310 T _xxx  
Found in /usr/lib/libyyyy.a
```

ここで"Found in "の右側が、関数xxxを定義しているライブラリへのパスである。見つかったライブラリ名をlibyyyy.aとすると、GCCのコマンド行に指定すべきオプションは-lyyyyである:

```
gcc -o test test.c -lyyyy
```