

x86以外のアセンブラ入門

愛甲健二

自己紹介

- 愛甲健二
- 26歳
- 独身
- 引きこもり
- asm歴4年くらい

目次

- アセンブラを使う人たちとは？
- CPUは友達だ！
- これからの演算装置とアセンブラ

第一章

- アセンブラを使う人たちとは？
- CPUは友達だ
- これからの演算装置とアセンブラ

どんな人がアセンブラをやっている？

- OS作っている人？
- VM作っている人？
- マルウェアを解析している人？
- 脆弱性を探す人？
- 無料でシェアウェアソフトを使う人？
- 楽しんでエンディングまで辿りつきたい人？

バックグラウンドはさまざま…

なぜアセンブラを学ぶの？

大抵は、

- どうしてもアセンブラでなければ解決できない問題にぶつかり仕方なく使用

でも稀に、

- もっと根本的なコア技術を知りたい
- アセンブルコード読んでると妙に落ち着く…
- アセンブラ…(*'Д`)ハアハア

というような方々もいる…

アセンブラって需要(就職)あるの？

- とても難しい質問です

google検索してみた

Google™ 求人 Java 検索 検索オプション
表示設定
 ウェブ全体から検索 日本語のページを検索

ウェブ [検索ツールを表示](#) 求人 Java の検索結果 約 2,560,000 件中 1 - 10 件目 (0.36 秒)

2,560,000件がヒット！

Google™ 求人 アセンブラ 検索 検索オプション
表示設定
 ウェブ全体から検索 日本語のページを検索

ウェブ [検索ツールを表示](#) 求人 アセンブラ の検索結果 約 66,300 件中 1 - 10 件目 (0.10 秒)

66,300件がヒット！

- 少なくともJavaよりは少ないみたいです
- でも、さすがに相手が悪いので、比較対象をCOBOLに

COBOLと比較してみた

Google™ [検索オプション](#)
[表示設定](#)

ウェブ全体から検索 日本語のページを検索

ウェブ [検索ツールを表示](#) 求人 Cobol に一致する日本語のページ 約 225,000 件中 1 - 10 件目 (0.21 秒)

225,000件がヒット！

Google™ [検索オプション](#)
[表示設定](#)

ウェブ全体から検索 日本語のページを検索

ウェブ [検索ツールを表示](#) 求人 アセンブラ に一致する日本語のページ 約 68,300 件中 1 - 10 件目 (0.23 秒)

68,300件がヒット！

- 結果、アセンブラは、**COBOLの1/3程度**の求人数

ちなみに、こういう記事も

市場価値の高い言語はどれだ？ 求人数は
Java、年収はC#がトップ ワークポートが調査<http://www.atmarkit.co.jp/news/200902/13/wp.html>

	言語	2008	2007
1位	JAVA	683	783
2位	C	653	764
3位	C++	345	314
4位	PHP	272	270
5位	C#	223	172

プログラミング言語別 求人件数ランキング(単位:件)

結論、つまりアセンブラやる人って

- とりあえず、求人とか年収とかどうでもいい
- COBOLの1/3程度の需要でも気にしない
- お金にならなくても気にしない
- きっと大切なのはそれが楽しいかどうかです

もしかしたら、

ただ純粹にコンピュータが好きだけの人たちなのかも…

今回はそういう人たちのお話です

第二章

- アセンブラを使う人たちとは？
- CPUは友達だ！
- これからの演算装置とアセンブラ

CPUとは

- 僕らが一番仲良くなりたい物理デバイス
- でも言葉があまり通じない
- だから彼ら(彼女ら)の言葉を学ぶんだ！

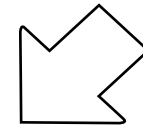
もっとも身近にいるCPU x86

- Intel社が開発したマイクロプロセッサでx86系と呼ばれる
- 主に汎用機(パソコン)で使用されている
- いろんなことを、そつなくこなす話の分かるやつ

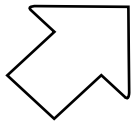
x86 (architecture)

- Intel社が開発したマイクロプロセッサシリーズ
- パソコンのCPU市場をほぼ独占
- 汎用性が高い

適当なelfファイルをgdbで読むと…



```
cmp    al, 8Bh
jz     short .exit
```



```
movb $0x41, 0xffffffff7(%ebp)
addl $0x20, %eax
```

適当なexeファイルをIDAProで開くと…

でもコンピュータはPCだけじゃない

- 携帯電話が登場して10年以上経過しています
- ゲーム機だって映画なみのクオリティで3D演算します
- TVにもCPUが搭載される時代です

というわけで、たまにはPC以外にも目を向けてみたり…

PLAYSTATION 3

- SCEが開発した次世代ゲーム機
- BD (Blu-ray) とHDMI端子を採用し、フルハイビジョンにも対応
- SCE、IBM、東芝が共同で開発したマイクロプロセッサであるCellを搭載し、次世代のコンピュータとして発売
- FF13も発売予定

様々な謳い文句がささやかれるが、僕らとしては…

何よりもまずはCellに興味深々

- Blu-rayとかHDMIとかゲームタイトルはとりあえず置いておいて、まずはCellを触ってみたい
- でもどうやってCellプログラミングするの？

Ubuntu Linux on PS3

- PS3用のUbuntu Linuxイメージが配布されている
- この辺りもだいぶ整備されており、インストールも難しくない
- PS3も現在は4万円程度で購入できる(ネットブック並)

というわけで、さっそくCellアセンブラをみってみる…

Cellのコードをデバッグ

```
$ cat >w.c
#include <unistd.h>

int main(void)
{
    write(1, "Hello PS3¥n", 10);
    return 0;
}
^C
$ gcc -Wall -static w.c -o w
$ ./w
Hello PS3
$ gdb w
(gdb) disassemble main
Dump of assembler code for function main:
0x1000047c <main+0>: stwu r1, -32(r1)
0x10000480 <main+4>: mflr r0
0x10000484 <main+8>: stw r0, 36(r1)
```

```
0x10000488 <main+12>: stw r31, 28(r1)
0x1000048c <main+16>: mr r31, r1
0x10000490 <main+20>: li r3, 1
0x10000494 <main+24>: lis r9, 4096
0x10000498 <main+28>: addi r4, r9, 2252
0x1000049c <main+32>: li r5, 10
0x100004a0 <main+36>: bl 0x10008cf0
0x100004a4 <main+40>: li r0, 0
0x100004a8 <main+44>: mr r3, r0
0x100004ac <main+48>: lwz r11, 0(r1)
0x100004b0 <main+52>: lwz r0, 4(r11)
0x100004b4 <main+56>: mtlr r0
0x100004b8 <main+60>: lwz r31, -4(r11)
0x100004bc <main+64>: mr r1, r11
0x100004c0 <main+68>: blr
End of assembler dump.
(gdb)
```

PowerPCやMIPSに似てる？

システムコール呼び出しの確認

```
(gdb) disassemble write
Dump of assembler code for function write:
0x10008cf0 <write+0>: lwz    r10,-29824(r2)
0x10008cf4 <write+4>: cmpwi r10,0
0x10008cf8 <write+8>: bne-  0x10008d0c <write+28>
0x10008cfc <__write_nocancel+0>: li    r0,4
0x10008d00 <__write_nocancel+4>: sc
```

以上の結果から

- r3レジスタ以降に引数をセット
- r0レジスタにシステムコール番号をセット
- sc命令でシステムコールを実行

Cellプログラミング

```
$ cat >w2.s
.section .data
.LC0: .string "Hello PS3¥n"
.section .text
.globl main
main:
    li 0,4 ## write // r0 = 4
    li 3,1 // r3 = 1
    lis 9,.LC0@ha // r9 = .LC0@ha << 16
    la 4,.LC0@l(9) // r4 = r9 + .LC0@l
    li 5,10 // r5 = 10
    sc
    li 0,1 ## exit // r0 = 1
    li 3,0 // r3 = 0
    sc
^C
$ gcc w2.s -o w2
$ ./w2
Hello PS3
```

無事アセンブラで書けました＼(^o^)/

でもこれだと、ただのPowerPC？

そう思ったので、検索してみたら…

Cell はマルチコアCPUで、1つのCPUの中に9個のプロセッサコアをもつ。1個の汎用的なプロセッサコアと、8個のシンプルなプロセッサコアを組み合わせたヘテロジニアスマルチコア。汎用プロセッサコアはPowerPC Processor Element (PPE) と呼ばれ、8個のコアはSynergistic Processor Element (SPE) と呼ばれる。

Wikipediaより

と、書かれてる

というわけで、

さっそく、そのSPEとやらで動くアセンブラを確認

SPE用のアセンブラ命令

```
$ w.spu-gcc -S
$ cat w.s
.section .rodata
.LC0:
    .string "Hello PS3¥n"
.text
.global main
main:
    stqd    $lr, 16($sp)
    stqd    $sp, -32($sp)
    ai      $sp, $sp, -32
    il      $3, 1
    ila     $4, .LC0
    il      $5, 10
    brsl    $lr, write
    il      $2, 0
    ori     $3, $2, 0
    ai      $sp, $sp, 32
    lqd     $lr, 16($sp)
    bi      $lr
```

- PPEと比べ命令が違う
- SPEで実行されるため、gdbでデバッグできない
- 演算に特化しているためか、システムコールも呼べない(左のコードは実行不可)
- Wikipediaによると128ビットのレジスタを128個保持する
- なんか難しそうだ

Cellまとめ

- とりあえず、Linuxも動くし、gcc/gdbも動く
- アセンブラはPowerPC系
- SPEを使いこなせればまだまだ可能性大
- 今回は計算速度系のベンチマークはとってないが、Corei7とも比較してみたい

第三章

- アセンブラを使う人たちとは？
- CPUは友達だ！
- **これからの演算装置とアセンブラ**

CPUの進化の方向性

- 熱処理問題により、クロック数はすでに限界値
- 今後はコアを増やしての並列処理の方向へ
- でも単純にコアを増やしただけで処理は向上する？
- GPUといった専門的な演算を行う装置もある
- 専門的な演算装置も含めたマルチコア化

ASMなプログラマ的には

- 64ビット、128ビットって、現実問題、解析作業(リバースエンジニアリング)が相当困難になるのでは？
- レジスタ数が128個とか、もはや人間では管理できない？
- 今後、CPUが複雑化していき、最終的にはASMLレベルで作業を行うエンジニアがいなくなるかも
- でも、そういった不安も含めて興味はある
- それにPCの世界では、まだまだ32ビットが現役

まとめ

- 実際問題、アセンブラはあまり仕事ない
- でも、ここ数年のCPUの進化は面白いです
- CPU以外の演算装置も出現しています(GPUとか)
- 今後どうなるか分からないけど
- Webセキュリティ関連が一段落ついた今だからこそ、ASMとかどうでしょう？

おまけ

iPhoneについて

iPhone / iPod touch

- Apple社が開発したスマートフォン
- タッチパネルを主体とした次世代の携帯端末？
- 日本ではソフトバンクモバイルから販売されている
- 外からは見えないが内部ではMacOSXが稼働
- iPod touchはiPhoneから電話など一部の機能を省いたもの

多くのCMで日本でも言葉として一般化した

ARMアーキテクチャ

- ARM Ltdにより開発されている
- **低消費電力**を売りにした32ビットCPU
- モバイル機器によく搭載されている

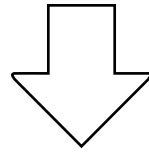
x86との違い

- 汎用レジスタが若干多い(15個くらい)
- PCレジスタ(EIP)が2つ先の命令を指す
- 命令長が4バイト固定
- 製品版のIDAProでなければ読めない

ARMアセンブラ

x86

```
0x00001fba <main+0>: push %ebp           // ebpをスタックへ  
0x00001fbb <main+1>: mov  %esp, %ebp // ebp = esp  
0x00001fbd <main+3>: sub  $0x10, %esp // esp = esp - 0x10
```



ARM

```
0x00001ee4 <main+0>: stmdb sp!, {r7, lr} // r7とlrをスタックへ  
0x00001ee8 <main+4>: add  r7, sp, #0 // r7 = sp + 0  
0x00001eec <main+8>: sub  sp, sp, #16 // sp = sp - 16
```

慣れればそれほど難しくなさそう？

Demo

ARMアセンブラやってみる

ご清聴有難うございました
m(_ _)m

Any Questions?