計算機システム Ⅱ ・第 1 回

2017 年 9 月 20 日

今回の内容

1.1 シラバス抜粋

講義概要	パソコンなどの計算機の標準的なハードウェアの仕組みを CPU の命令セットを中心に解説します。また、Windows、Linux、Mac OS X などのオペレーティングシステムの核となる部分 (カーネル) の役割と仕組みについて学びます。
到達目標	自分の書いた C プログラムが、なぜ動いてくれるのか、その仕組みを理解することが目標です。
講義方法	配布した資料に沿って講義を行います。
系統的履修	情報処理の基礎、計算機システム I の内容が前提となります。
成績評価の方法	期末試験 (100 点満点) と提出された演習課題等で評価します。期末試験が x 点、課題の得点率が y % のとき、総合的な成績は $x+(100-x)y/500$ 点 (端数切り捨て) となります。
講義計画	(1) プログラムが動くということ (2) CPU とメモリ (3) データ転送命令と演算命令 (4) 分岐命令 (5) キャッシュメモリ (6) パイプライン処理 (7) プログラミングへの応用 (8) オペレーティングシステムの役割 (9) カーネル (10) プログラムとプロセス (11) システムコール (12) ファイルシステム (13) メモリ管理 (14) ユーザープロセスへのメモリ割り当て (15) まとめ
テキスト	なし。配布資料は次の Web ページからから入手できます。 http://www602.math.ryukoku.ac.jp/%7Enakano/CSys2/
参考文献	矢沢久雄『プログラムはなぜ動くのか』(日経 BP) 2,592 円 デイビッド・ A ・パターソン、ジョン・ L ・ヘネシー『コンピュータの構成と設計 第 4 版 (上)』(日経 BP 社) 4,536 円 デイビッド・ A ・パターソン、ジョン・ L ・ヘネシー『コンピュータの構成と設計 第 4 版 (下)』(日経 BP 社) 4,536 円 大久保英嗣『オペレーティングシステムの基礎』(サイエンス社) 1,728 円

1.2 プログラムが動くということ

この科目の受講者なら、C 言語のプログラムを書いて、そのプログラムを実行してみたことがあるはずです。たとえば、次のような C プログラムを考えてみます。

myprog.c

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5     int x, y;
 6
 7     printf("x = ");
 8     scanf("%d", &x);
 9     printf("y = ");
10     scanf("%d", &y);
11     printf("%d + %d = %d\n", x, y, x+y);
12     return 0;
13 }
```

このような C プログラム myprog.c を作成して、たとえば情報処理実習室の Linux 環境で次のようにコンパイルすると、機械語プログラム myprog ができます。

```
$ cc -o myprog myprog.c
```

また、こうして作成した機械語プログラム myprog は、次のように起動して、myprog.c に書いた通りに計算機を動作させることができるわけです。

```
$ ./myprog
x = 7
y = 25
7 + 25 = 32
```

ソースプログラムを書いて、コンパイルし、実行する — この一連の手順は、これまで何度も繰り返して来たことですから、ここで紹介した流れの中で起きたことには、もう何の疑問を持たないかも知れません。しかし、よくよく考えてみると、まだよく分かっていない部分が残っていることに気づきます。

メモ

1.2.1 謎、その 1

CPU のごく基本的な仕組みは、2 年次前期開講の「計算機システム I」で勉強しましたが、そこでは、現実の CPU が持っているはずのいくつかの仕組みが省略されていました。たとえば、myprog.c では、キーボードから入力された整数値を知ったり、デイスプレイ上のあるウィンドウ内に文字列

を表示したりしているわけですが、この C プログラムをコンパイルしてできた機械語プログラムを実行している CPU は、どのようにしてキーボードの押されたキーを知ったり、ディスプレイの表示を変えたりすることができるのでしょうか。CPU とつながっているはずのメモリや入出力機器などとの関係がよく分かっていませんし、そのために CPU に備わっている (はずの) 仕組みを知りません。

メモ

1.2.2 謎、その 2

myprog.c をコンパイルすることで myprog という機械語プログラムができましたが、この myprog は 1 つのファイルとしてハードディスクなどの補助記憶装置に記憶されているはずです。このプログラムを CPU で実行するためには、ハードディスク上の機械語プログラムをメモリ上にコピーしないといけないはずですが、この仕事を行ったのは一体誰なのでしょうか。この仕事も CPU が行ったはずなのですが、その仕事はやはり機械語プログラムとしてどこかに存在しないといけないはずです。そのプログラムはどこから来たのでしょうか。

また、myprog.c で使われていた scanf や printf という関数も、結局は機械語プログラムのはずですが、これらのプログラムは一体どこにあって、いつどのように myprog とつながったのでしょうか。また、printf を呼び出すことで、ディスプレイ上のウィンドウには文字列が表示されたわけですが、この文字列はどのようにして表示されたのでしょうか。ディスプレイ装置自体にこの文字列を表示する仕組みが備わっているとは思えませんので、やはり何か隠れた機械語プログラムが働いていることが想像できますが、このプログラムはどこから来て、いつどのように myprog とつながったのでしょうか。

メモ

1.2.3 謎、その 3

私たちが書いた C プログラムはコンパイラと呼ばれるプログラム (先の例では cc コマンド) によって、機械語プログラムと呼ばれる CPU が直接実行できる形のプログラムに変換されますが、その「機械語」がどのようなプログラミング言語なのか知りません。機械語については、「計算機システム I」以前にも、1 年次後期開講の「情報処理の基礎」で少し紹介されていましたが、私たちが実際に使用している CPU の機械語がどのようなものであるかについては、まだ知らないままです。

情報処理実習室の Linux 環境で、先の `myprog.c` という C プログラムをコンパイルすると、この資料の付録にあるような機械語プログラムに変換されますが、このプログラムに現れている機械語命令は、それぞれどんな意味を持っているのでしょうか[1]。

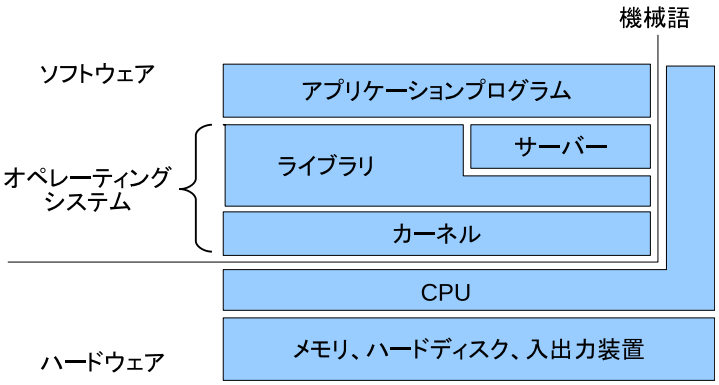
メモ

1.3 この科目で学ぶこと

この科目は、これらの謎に (少しでも) 答えることを目標としています。「謎、その 1」は、CPU を中心とした計算機のハードウェアの仕組みに関する謎です。計算機の (主にハードウェアの) 仕組み (構成) やその設計を「計算機アーキテクチャ」と呼びます。この科目の内容の半分は、この計算機アーキテクチャについてです。計算機アーキテクチャは、まず、全体として必要な仕事ができること、また、それを可能限り効率的に (たとえば高速に) 行うことができるように作られます。

「謎、その 2」は、オペレーティグシステムと呼ばれるソフトウェアに関する謎です。「情報処理の基礎」では、そのほんの「さわり」だけを勉強したのですが、この科目では、もう少しその詳細に立ち入ることになります。計算機アーキテクチャはオペレーティングシステムがうまく機能できるようになっていなければなりませんので、この 2 つには密接な関係があります。

「謎、その 3」は、オペレーティグシステムを含むソフトウェアと、ハードウェア (計算機アーキテクチャ) のつなぎ目に関する謎ということになります。ハードウェア側からみると、機械語をどのように定めるか (どのような機械語命令を用意するか) は計算機アーキテクチャに関する問題となりますし、ソフトウェア側からみると、それをいかに効果的に利用するかという問題となります。



この科目では、これらの 3 つの内容を、主にアプリケーションプログラムを作成する際の視点で勉強していきます。

[1] さらに言えば、コンパイラはどのような仕組みでこの変換をすることができたのかについても知らないままです。この仕組みは、3 年次前期に開講される「記号処理」という科目で勉強することができます。

1.4 付録： myprog.c をコンパイルして得られた機械語プログラム

下のプログラムは、情報処理実習室の Linux 環境で、objdump コマンドを

```
$ objdump -d myprog
```

のように実行して、機械語プログラム myprog をディスアセンブル (disassemble)[2] して得られたものの一部です。各行には、16 進表記のメモリ番地と「:」、その番地から置かれた機械語命令のバイト列 (16 進表記)、その機械語命令のニーモニック (機械語命令の意味を人間が理解しやすいように英文字や記号、数字の並びで表したもの) が書かれています。

```
アドレス   機械語命令のバイト列    ニーモニックで表した機械語命令
40059d:  55                   push   %rbp
40059e:  48 89 e5             mov    %rsp,%rbp
4005a1:  48 83 ec 10          sub    $0x10,%rsp
4005a5:  bf a4 06 40 00       mov    $0x4006a4,%edi
4005aa:  b8 00 00 00 00       mov    $0x0,%eax
4005af:  e8 bc fe ff ff       callq  400470 <printf@plt>
4005b4:  48 8d 45 f8          lea    -0x8(%rbp),%rax
4005b8:  48 89 c6             mov    %rax,%rsi
4005bb:  bf a9 06 40 00       mov    $0x4006a9,%edi
4005c0:  b8 00 00 00 00       mov    $0x0,%eax
4005c5:  e8 d6 fe ff ff       callq  4004a0 <__isoc99_scanf@plt>
4005ca:  bf ac 06 40 00       mov    $0x4006ac,%edi
4005cf:  b8 00 00 00 00       mov    $0x0,%eax
4005d4:  e8 97 fe ff ff       callq  400470 <printf@plt>
4005d9:  48 8d 45 fc          lea    -0x4(%rbp),%rax
4005dd:  48 89 c6             mov    %rax,%rsi
4005e0:  bf a9 06 40 00       mov    $0x4006a9,%edi
4005e5:  b8 00 00 00 00       mov    $0x0,%eax
4005ea:  e8 b1 fe ff ff       callq  4004a0 <__isoc99_scanf@plt>
4005ef:  8b 55 f8             mov    -0x8(%rbp),%edx
4005f2:  8b 45 fc             mov    -0x4(%rbp),%eax
4005f5:  8d 0c 02             lea    (%rdx,%rax,1),%ecx
4005f8:  8b 55 fc             mov    -0x4(%rbp),%edx
4005fb:  8b 45 f8             mov    -0x8(%rbp),%eax
4005fe:  89 c6                mov    %eax,%esi
400600:  bf b1 06 40 00       mov    $0x4006b1,%edi
400605:  b8 00 00 00 00       mov    $0x0,%eax
40060a:  e8 61 fe ff ff       callq  400470 <printf@plt>
40060f:  b8 00 00 00 00       mov    $0x0,%eax
400614:  c9                   leaveq
400615:  c3                   retq
```

[2] ニーモニックなどで書かれたプログラムを機械語プログラムに変換することをアセンブル (assemble) する、その逆をディスアセンブルする (disassemble) と言います。

1.5　付録：16 進表記

ディジタル計算機では、そこで扱う情報をすべてビット列 (0/1 の列) として表現しますので、機械語プログラムや計算機アーキテクチャの話をする際には数十桁の 2 進数が頻繁に登場することになります。このような 2 進表記は人間にとっては扱い難いので、下位の桁から 4 桁 (4 bit) ずつひとまとめにして、その 4 bit のビットパターンを下の表の 16 種類の記号でそれぞれ表し、その記号を同じ順に並べて、全体を 16 進表記で表すことがよく行われます。

たとえば、0010 1110 というビットパターン (2 進表記) を 2e と、1100 0101 0001 0110 1101 1010 1000 0111 というビットパターン (2 進表記) を c516da87 と 16 進表記します。また、16 進表記であること明示するために、それぞれ、0x2e や 0xc516da87 のように先頭に 0x (あるいは 0X) を付けたり、2eh や c516da87h のように末尾に h (あるいは H) を付ける慣習もあります。

ビットパターン	記号
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	a
1011	b
1100	c
1101	d
1110	e
1111	f

この表では、1010 から 1111 までのビットパターンを英小文字 a から f に対応させていますが、小文字の代りに大文字 A から F を使用することもあります。4 桁の 2 進表記を 10 進表記した時、0 から 9 までは、そのままの数字で、10 から 15 までは、a から f までの英文字で表していることに注意してください。

計算機システム Ⅱ ・第 1 回・終わり