コンピューターのメモリーとポインター

山本昌志*

2007 年 2 月 9 日

概 要

コンピューターのメモリーの役割と動作概要を示す．メモリーのアドレスを理解した後，ポインターの基礎的なことを学ぶ．

1 前回の復習と本日の内容

1.1 前回の復習

前回は位取り記数法の学習をした．そこでは，2 進数，10 進数，16 進数の相互の変換方法を学んだ．これらの変換には十分慣れる必要がある．

1.2 本日の内容

本日の内容は，教科書 [1] の p.270–277 である．教科書で記述していない項目も含むが，理解して欲しい内容である．本日のゴールは，以下のとおりである．

- CPU とメモリーの関係が分かる．そして，メモリーのアドレスとデータの関係が分かる．
- ポインター変数はアドレスを格納するものである—ということが分かる．
- アドレス演算子で変数のアドレスを取り出し，間接参照演算子ではポインターが指し示す変数の値を取り出す—ということが分かる．

2 コンピューターの仕組み

2.1 コンピューターの基本構成

コンピューターは複雑な装置であるが，図 1 のような機能の集まりに分解できる[1]．それぞれの機能 (装置) は，制御信号線とデータ信号線で接続されている．そこを，0 と 1 のパルスの信号が信じられないくら

*独立行政法人　秋田工業高等専門学校　電気情報工学科

[1]複雑なものを機能毎に分解するのは，プログラムのサブルーチン (関数) の考え方とおなじである．複雑なものは，このようにモジュール単位に分割して考えるのは常套手段である

い高速でかつ調和を取って流れ，全体としてコンピューターが動作する．大量の信号がひとつも間違いなく伝送されるのは驚きである．この信号線は人間で言えば神経のようなものである．

コンピューターの基本構成は図1に示したとおりであるが，**制御装置**と**演算装置**，**記憶装置**，**入力装置**，**出力装置**を五大装置と呼ぶ．それぞれは，次のような働きがある．

制御装置 主記憶装置 (メインメモリー) に格納されているプログラム (命令) を受け取り，それを解読し，電気信号に変え，各装置に指令を出す.

演算装置 主記憶装置 (メインメモリー) に格納されているデータを受け取り，制御装置の指令に従い，それを加工 (処理) する.

記憶装置 記憶装置は，主記憶装置と補助記憶装置がある．

- **主記憶装置** 命令とデータからなるプログラムを格納する．
- **補助記憶装置** 主記憶装置には容量に制限があるので，それを越えるものをここに格納する．また，主記憶装置は電源を切ると，データが失われるので，半永久的に残したい場合，補助記憶装置に格納する．ハードディスクや CD-ROM がこれに当たる．

入力装置 コンピューターの外部からデータを取り込む装置である．キーボードやマウス等がこれに当たる．

出力装置 主記憶装置に格納されているデータをコンピューター外部に出力する装置である．ディスプレイやプリンター等がこれに当たる．

主記憶装置 (メインメモリー) は略して，メモリーと呼ばれることが多い．この講義でもメモリーといえば，主記憶装置 (メインメモリー) を指す．パソコンの広告を見ると「メモリー 512Mbyte 搭載」とか書いてある．これがまさにメモリーの大きさを示している．

これらのなかで，制御装置と演算装置をまとめて中央制御装置 (CPU:Central Processing Unit) と言う．とくに，これらを一つのチップにまとめたものを MPU(Micro Processing Unit) と言う．Intel 社の Pentium や AMD 社の Athlon，IBM 社の PowerPC などが MPU である．世間一般では MPU と CPU はほとんど同義語として使われるので，本講義では全て CPU に統一する．その方が諸君もなじみ深いであろう．

図1では，入力装置と出力装置は分けたが，ひとつで2つの機能を果たすものもある．例えば，LAN カードがそれにあたる．

2.2 メモリーと CPU

ここで，コンピューターを構成する最小の部品を考える．コンピューターは CPU とメインメモリーで動作する．メインメモリーにプログラムを格納して，CPU とデータの受け渡しを行い，データを処理することができる．図2のようなものである．ここに書いているデータバスとかアドレスバスについては，後で説明する．

プログラマーは，メモリーの内容をある程度自由に変更ができる．そのようなことから，メモリーを意識してプログラムを作成することが重要である．アセンブラー言語を使うとなると CPU についても意識が必要であろうが，C 言語ではそこまで要求しない．諸君は，メモリーを意識しろ!!!

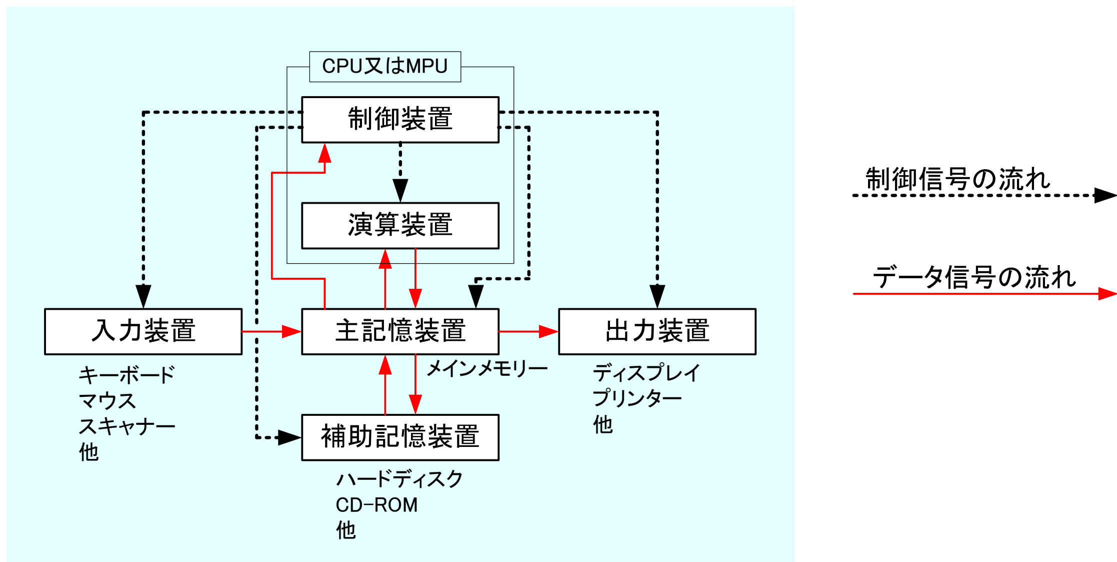

図 1: コンピューターの基本構成

2.3 メインメモリーのモデル

メインメモリーのハードウェアーの構造については，ここではどうでも良い．それよりか，メインメモリーのモデルを理解することが重要である．

メインメモリーの役割は，命令とデータからなるプログラムを記憶することである．記憶する内容は，0と1のビットパターンのみである．なぜならば，プログラムは全て，0と1の数字で表せ，二進数で表現可能であるからである．コンピューター内部では，データも命令も全て0と1の二進数で表現している．命令も二進数である;どうだ，驚いたか．ここでは，この二進数のビットパターンを格納するメモリーについて学習する．

プログラムという情報は記憶するだけでは全く役に立たない．記憶した内容を取り出せて初めて，活用ができる．また，格納できる情報がひとつでは全く役に立たない．多くの情報が格納でき，必要なときにいつでも取り出すことができる装置がメモリーである．

メモリーは大量の情報を格納し，その中の任意の情報を取り出すことができる．そのために，メモリーには記憶する場所毎に住所が決められている．この住所のことをアドレス (address) と言い，整数の番地がふってある．諸君が使っているパソコンのアドレスは32ビットで表現されている[2]．そして，一つの番地には，8個の0と1が記憶できる．この様子を図3に示す．

図を見て分かるとおり，2進数の表現は桁数が多くて人間にとって大変である[3]．そこで，通常は，2進数の4桁をまとめて，16進数で表す．そうすると，アドレスは16進数8桁，記憶内容は16進数2桁で表すことができ，分かりやすくなる．その様子を図4に示す．

ついでに述べておくが，1個の0あるいは1の情報量を1ビットと言う．8ビットで1バイトと言う．従っ

[2] CPUによりアドレスの表現は異なり，32ビットではないものもある．
[3] コンピューターにとっては，桁数が多くても，二進数の方が取り扱いやすい

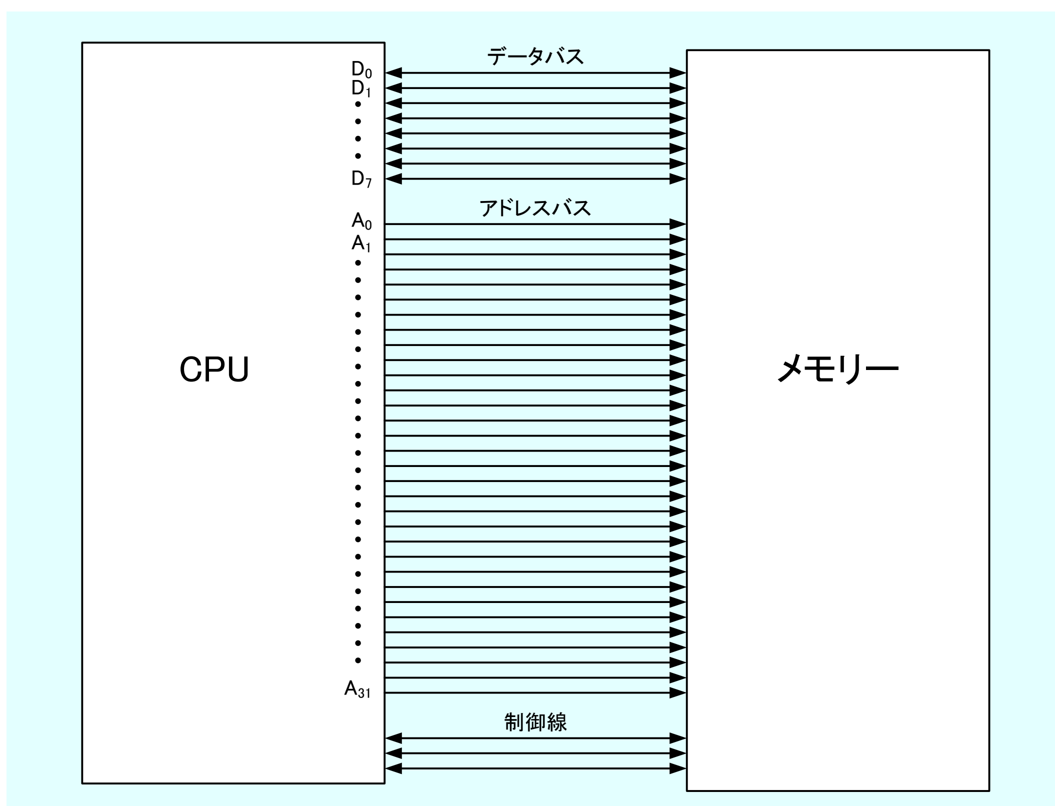

図 2: もっとも原始的なコンピューター．バスの矢印は情報の流れを示す.

て，メインメモリーの一つの番地 (アドレス) には，1 バイト (8 ビット) の情報が記憶できる．
メモリーについて覚えておくことは，以下の通りである．

- アドレスは 32 ビットで表現している．これは 16 進数では 8 桁である．
- ひとつのアドレスに 8 ビット (1 バイト) 記憶できる．
- 8 ビットを 1 バイトと言う．

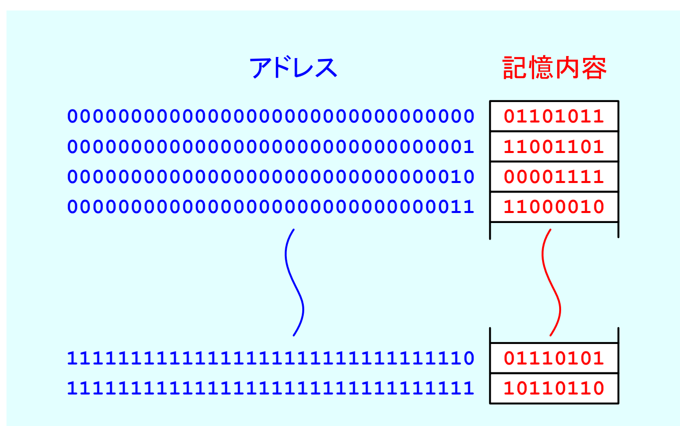

図 3: メモリーのモデル (2 進数)

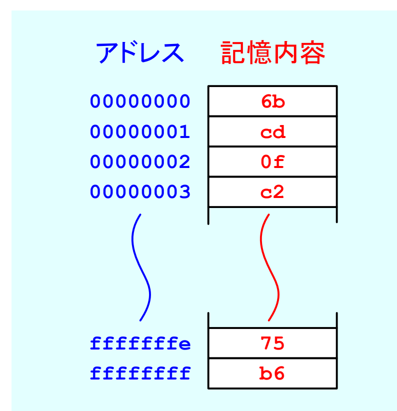

図 4: メモリーのモデル (16 進数)

ところで，CPU は，どのようにしてメモリーに情報を書き込んだりメモリーから情報を読み込んだりするのだろうか? 簡単にその仕組みを説明するので，図 2 のアドレスバスとデータバスを見よ[4]．CPU は読み書きを行いたいアドレスをアドレスバスの電圧を使って，メモリーに知らせる．32 本のアドレスバスの電圧を high—1 の状態— と low—0—の状態にすることにより，32 ビットのアドレスを指定する．すると，8 本のデータバスをとおして，データが流れるのである．読み込み/書き込みの区別は，CPU が制御線の電圧で決め，それに応じてメモリーが動作する．

2.4 メモリー中の変数の値

2.4.1 変数のバイト数

C 言語では，変数の型により，データのサイズが異なる．諸君が使っているパソコンの型毎のデータのバイト数は表 1 の通りである．文字型であれば，ひとつのアドレス内に格納することができる．ひとつのアドレスには 1 バイトのデータが格納でき，文字型のデータは 1 バイトの情報を持つからである．整数型では 4 つ，倍精度実数型では 8 個のアドレスが必要である．ひとつのアドレスにひとつのデータが記憶されている訳ではない．
付録のリスト 4 にデータ型のバイト数を調べるプログラムを載せておく．

[4]実際の CPU ではアドレスバスとデータバスの本数はこの図と異なる．

表 1: 変数の型とバイト数

型名	データ型	バイト数	ビット数
文字型	char	1	8
整数型	int	4	32
倍精度実数	double	8	64

2.4.2 メモリー中のビットパターン

それでは，実際にメモリーにデータが格納する様子を見よう．次のようにプログラムに書いたとする．

```
double x=-7.696151733398438e-4;
int i=55;
char a='a';
```

それぞれのデータは，

- x の値のビットパターンは，前回の講義の付録を見れば分かる．ただし，ちょっと難しい．分からなくてもよい．
- $(55)_{10} = (37)_{16}$ から i のビットパターンは分かる．
- 文字の'a' はアスキーコードの $(61)_{16}$ である．

となっている．実際にこれを確かめるプログラムを，付録のリスト 5 に示している．このプログラムを私のパソコンで実行させると，図 5 のようなメモリー配置になっていることが分かった．表 1 の通り，整数型と実数型は複数のアドレスにわたってデータが格納されていることが分かる．

鋭い学生は，データの並びが逆であることが分かるであろう．例えば，整数の i であるが，$(55)_{10} = (00000037)_{16}$ なので，$00 \to 00 \to 00 \to 37$ と並ぶと考えるだろうが，実際は図 5 のように逆に並ぶ．これは CPU がそのように作られているからである．このように逆に配置させる方法を**リトルエンディアン**と言う．Intel 社の CPU はリトルエンディアンである．一方，そのままのメモリーに配置する方法は**ビッグエンディアン**と呼ばれる．ここで述べたように，メモリーにデータを並べる方法は 2 通りあって，それをバイトオーダーと言う．どちらがより良いかは分からない．

ここで，理解しておくべきことは，以下の通りである．

- 必要なバイト数のメモリー領域を使いデータは格納される．

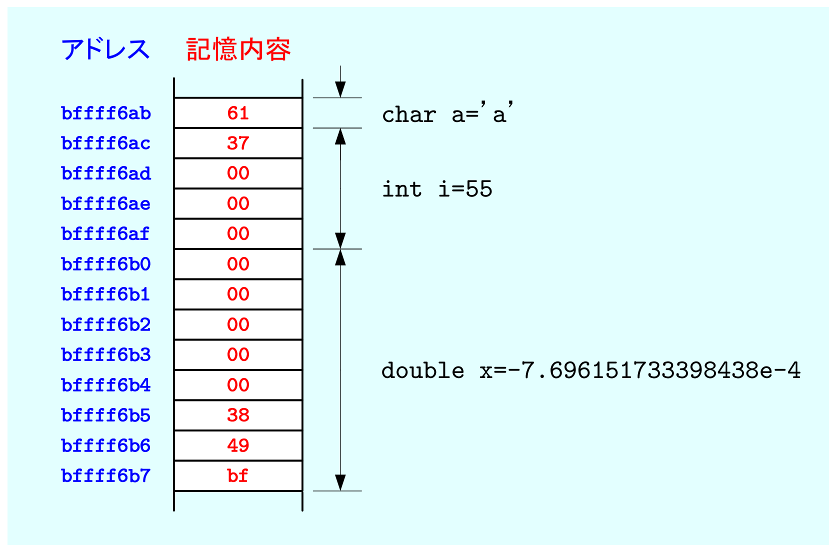

図 5: メモリー中に格納されたデータの例

2.5 プログラムが格納される様子

これまでは，メモリーのデータの格納方法を学習した．まえに，プログラム (命令とデータ) は全てメモリーに格納されると述べた．ここでは，もう少し進んで，プログラムがメモリーの中にどのように格納されているか調べてみよう．この辺のことと，マシン語が分かると，ハッカー (クラッカーと言った方が適切かも) になれるかも $\cdots$

それでは，リスト 1 に示す簡単なプログラムで，データとメモリーの格納アドレスを調べてみよう．このプログラムの各行の動作は，以下の通りである．まだ，詳細は分からなくても良いが，大体の流れをつかんで欲しい．

- 12 行 `main` 関数が格納された先頭アドレスを表示．関数名は，関数のプログラムが格納されている先頭アドレスを示している．それは，変換指定子%p でディスプレイに表示することができる．
- 13 行 関数 `func` が書かれている先頭アドレスを表示．
- 14 行 変数 `i` の先頭アドレスを表示．変数名に& を付けると，その先頭アドレスを取り出すことができる．&はアドレス演算子である．
- 26 行 変数 `i` の先頭アドレスを表示．
- 27 行 変数 `j` の先頭アドレスを表示．

リスト 1: メモリーのアドレス調査

```
1  #include <stdio.h>
2  int func(int i, int j);
3
```

```
4  /*======================================================*/
5  /*    メイン関数                                        */
6  /*======================================================*/
7  int main(void)
8  {
9    int i;
10
11   printf("--- address -----------\n");
12   printf("\tmain\t%p\n", main);
13   printf("\tfunc\t%p\n", func);
14   printf("\tmain-i\t%p\n",&i);
15
16   i=func(5,3);
17
18   return 0;
19 }
20 /*======================================================*/
21 /*  func 関数                                           */
22 /*======================================================*/
23 int func(int i, int j)
24 {
25
26   printf("\tfunc-i\t%p\n",&i);
27   printf("\tfunc-j\t%p\n",&j);
28
29   return i*j;
30 }
```

実行結果

```
--- address -----------
        main    0x8048368
        func    0x80483eb
        main-i  0xbffff6b4
        func-i  0xbffff690
        func-j  0xbffff694
```

実行結果から，命令とデータは図 6 のようになっていることが分かるであろう．命令である関数は，大体近くのメモリの先頭付近に配置されている．しかし，データの内容を格納する変数は，ずっと離れてメモリーの後ろの方に割り当てられている．

関数の中で宣言される変数は，ローカル変数と言い，その宣言した関数でのみアクセスが可能である．従って，同じ名前であるが，違う関数で宣言されたローカル変数は全く別物である．図 6 で分かるように，関数 `main` と関数 `func` で同じ名前のローカル変数 `i` を宣言してるがメモリー上の配置は全く異なる．このことからも，名前は同じであるが，全く違うものであることが理解できる．

ここで，理解しておくべきことは，以下の通りである．

- プログラムは命令とデータから構成され，いずれもメモリーの中に格納される．
- プログラムの関数 (これが命令) が格納されるアドレスは，関数名で参照できる．
- データが格納されるアドレスは，変数名の前に & を付けることで参照できる．& はアドレス演算子である．
- アドレスの表示には変換指定子 %p を使う．
- ローカル変数は名前が同じでも，メモリーの配置場所は異なる．正確言うと，その関数が呼び出されたときのみ，ローカル変数はメモリーに割り当てられる．

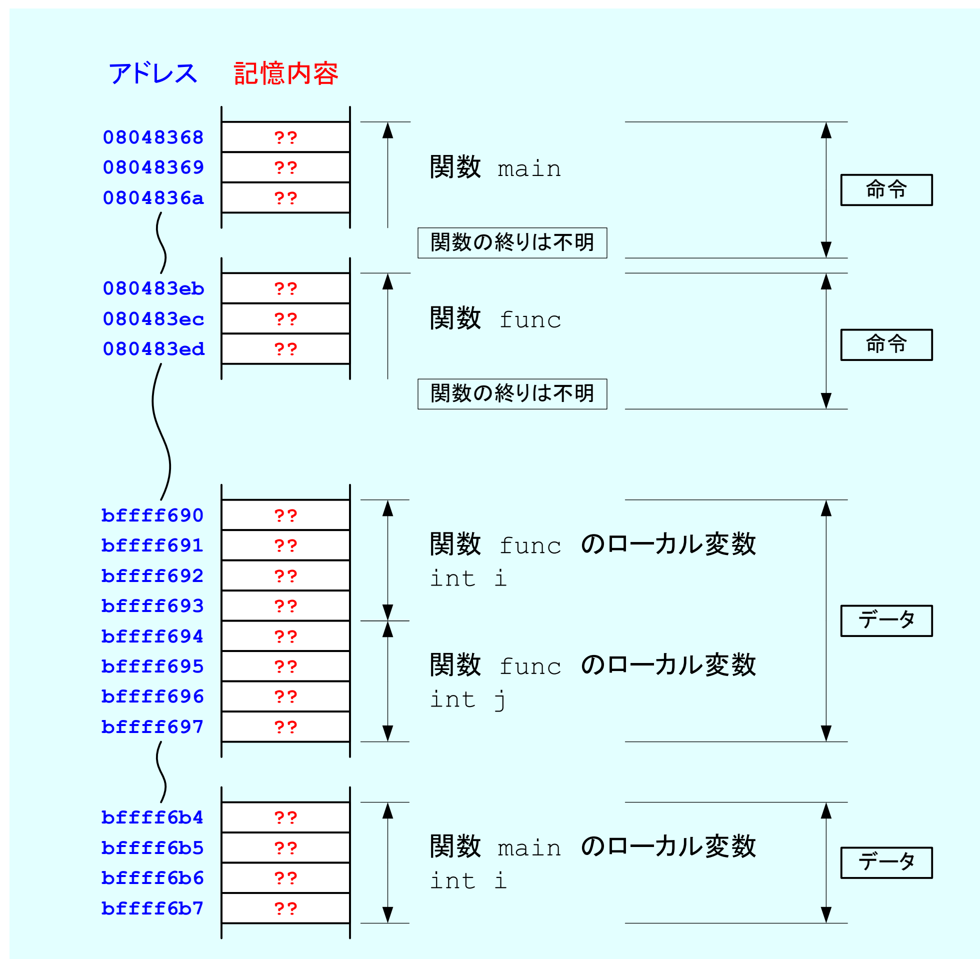

図 6: プログラムのメモリーへの格納．記憶の内容は不明なので，??としている．

3　ポインターとなにか

3.1　ポインター変数と演算子

3.1.1　ポインター変数

これまでの話で，メモリーというものが大体分かったと思う．そして，その内容とともに，アドレスが重要であることも分かったであろう．あるいは，アドレスを上手に操作すれば，いろいろなこと (悪いことも) ができそうだ—ということも分かったであろう．

アドレスを操作するとなると，アドレスを入れる変数が欲しくなる．2.3 節で述べたように，アドレスは 32 ビットである．また，`int` 型のデータも 32 ビットである．従って，`int` 型の変数にアドレスを入れることがあできそうである．具体的には，`hoge` と言う変数のアドレスを `int` 型の変数 `i` に，次のような文で，

```
i=&hoge;
```

と代入する．しかし，これはコンパイラーにより警告が出され，推奨される方法でない[5]．たまたま，私が使っているコンパイラーでは警告で済んでいるが，エラーを出すものもあるであろう．そもそも，アドレスのビット数と `int` 型のビット数が同じであるのは偶然にすぎない．

幸いなことに，C 言語にはアドレスを格納する仕組みが用意されている．アドレスを格納するための変数を**ポインター変数**[6]という．それは，

```
int *pi;
double *px;
```

と宣言する．アスタリスク (*) をつければ，ポインター変数の宣言になる．

3.1.2　アドレス演算子

変数のアドレスは，**アドレス演算子** (`&`) により取り出すことができる．たとえば，整数型変数 `i` と実数型変数 `x` のアドレスは，`&i` と`&x` とすると取り出すことができる．ここで，間接参照演算子`&`は，それに引き続く変数の先頭アドレスを取り出す演算子である．取り出したアドレスは，ポインターに

```
pi=&i;
px=&x;
```

のようにして代入できる．アドレス演算子 (`&`) により変数の先頭アドレスを取り出して，代入演算子 (`=`) を用いて，ポインター変数に代入している．

3.1.3　間接参照演算子

ポインター機能は，アドレスの格納のみに止まらず，そのアドレスが示しているデータの内容も表すことができる．今までの例の通り，ポインター変数には変数の先頭アドレスが格納されている．そして，ポインターの宣言の型から，そのポインターが指しているデータの内容までたぐり寄せることができる．ポインター `pi` と `px` が示しているデータの値を，整数型変数 `j` と実数型変数 `y` に代入する場合

[5]キャスト (強制型変換) を使って警告を消すこともできるが邪道である．
[6]たんに,「ポインター」と呼ぶことの方が多い．

```
j=*pi;
y=*px;
```

と書く．ここで，アスタリスク (*) は**間接参照演算子**[7]で，ポインターが示しているアドレスのデータを取り出す演算子である．このようにアドレスのみならず，そのアドレスのデータの型までポインターは持っているから，これが可能なのである．このことから，アドレスとは言わずにポインター (pointer 指し示すもの) と言うのであろう．

まったくもって，紛らわしいことに，間接参照演算子と積の演算子は同じアスタリスク (*) をつかう．C 言語の悪いところだが，そうなってしまっているので仕方ない．多分，コンパイラーは前後の式からどちらなのか判断しているのだろう．

- ポインター変数は，アドレスを格納する変数である[a]．
- ポインターの宣言には，型名とアスタリスク (*) を付ける．
- 変数のアドレスを取り出すには，変数名の前にアンパサンド (&) をつける．&はアドレス演算子である．
- ポインターが示しているデータの値を取り出すためには，ポインター変数の前にアスタリスク (*) をつける．*は間接参照演算子である．

[a] 正確に言うとちょっと違うが，ほとんど正しい．また，アドレスはメモリーの物理的なアドレスではなく，仮想アドレスである．この辺のところは余り気にしないことにする．

3.2 プログラム例

リスト 2 に示すプログラムで，ポインターの意味とそれに関わる演算子の動作の基礎的なことを理解しよう．このプログラムの各行の内容は，以下の通りである．1 行毎にきっちり理解することが重要である．

5 行 整数型のポインター変数 p を宣言している．変数 p は整数型のデータの先頭アドレスを格納する．

8 行 変数 i の先頭アドレスをアドレス演算子&により取り出し，ポインター変数 p に代入．

10 行 整数型変数 i の先頭アドレスを変換指定子%p により表示している．

11 行 ポインター変数 p の先頭アドレスを変換指定子%p により表示している．

13 行 整数型変数 i の値を 16 進数で表示している．変換指定子%0x を指定するとデータは 16 進数表示になる．

14 行 ポインター変数 p の値を 16 進数表示の変換指定子%0x により表示している．ただし，ポインターはアドレスなので，強制型変換 (キャスト) により，符号なし整数にしている[8]．

16 行 ポインターが指し示すアドレスに格納されているデータを表示している．

[7] 教科書では,「間接演算子」と記述している．どちらでも同じである．ただし，間接参照演算子と言う方が詳しく意味を説明しているので，ここではこちらを使う．

[8] 強制型変換しなくても実行は可能であるが，コンパイル時に型の不一致の警告がでる．

リスト 2: アドレスをポインターに代入して，変数のアドレスと内容を検査

```
 1 #include <stdio.h>
 2
 3 int main(void)
 4 {
 5   int *p;
 6   int i=0x11223344;
 7
 8   p=&i;
 9
10   printf("address i %p\n", &i);
11   printf("address p %p\n", &p);
12
13   printf("value i %0x\n", i);
14   printf("value p %0x\n", (unsigned int)p);
15
16   printf("value *p %0x\n", *p);
17
18   return 0;
19 }
```

実行結果

```
address i 0xbffff6b0
address p 0xbffff6b4
value i 11223344
value p bffff6b0
value *p 11223344
```

この実行結果から，メモリーは図 7 のようになっていることが分かる．ポインター p には，整数変数 i の先頭アドレスが格納されている．さらに，ポインター p に間接参照演算子*を作用 (*p) させることにより，ポインターが指し示すアドレスの内容を取り出している．また，どんな変数でも，アドレス演算子&で，メモリーのアドレスが取り出せている．これらのことをしっかり理解すると，ポインターは難しくない．

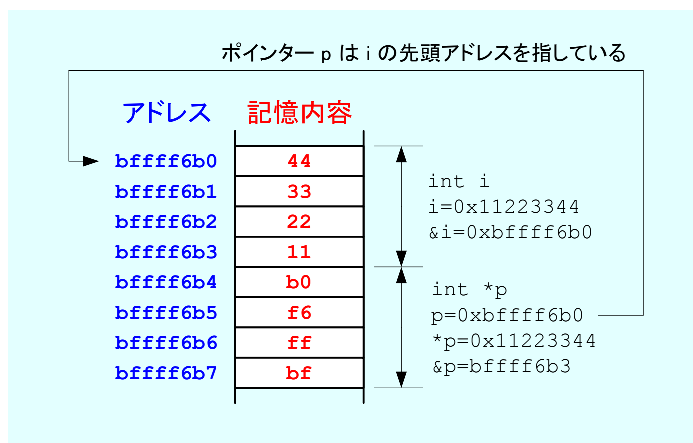

図 7: リスト 2 のプログラム実行後のメモリーの内容

3.3　ポインターに関係する演算子

ポインターに関係する演算子を表 2 にまとめておく．ただし，各変数は

```
double x, *xp;
```

と宣言したとする．

表 2: 普通の変数とポインター変数に演算子を作用させた場合に取り出せる値．&:アドレス演算子．*:間接参照演算子

演算子	通常の変数 (x)	ポインター変数 (xp)	例
無し	格納されている値	格納されているアドレス	x,xp
&	変数のアドレス	ポインター変数のアドレス	&x,&xp
*	コンパイルエラーのため不可	ポインターが示すアドレスに格納されている値	*xp

まとめると，重要なことは以下の通りである．

- 通常の変数には値が，ポインターにはアドレスを格納する．
- アドレス演算子&は，それに引き続く変数 (ポインター変数も含む) のアドレス返す．
- 間接参照演算子*は，それに引き続くポインター変数が指し示すアドレスの値を返す．

4　プログラム作成の練習

[**練習 1**]　以下のように動作するプログラムを作成せよ．

1. 整数型の変数を用意する．それに，整数 123456 を格納する．
2. 整数型の変数の値を表示させる．
3. 整数型の変数の先頭アドレスを表示させる．

[**練習 2**]　以下のように動作するプログラムを作成せよ．

1. 整数型の変数を用意する．それに，整数 13579 を格納する．
2. 整数型のポインターを用意する．それに，先ほど用意した整数型の変数の先頭アドレスを格納する．
3. 整数型のポインターが指し示す値—整数 13579—を表示させる．

[**練習 3**]　以下のように動作するプログラムを作成せよ．

1. 整数型の変数を用意する．それに，整数 24680 を格納する．
2. 整数型のポインターを用意する．それに，先ほど用意した整数型の変数の先頭アドレスを格納する．
3. 整数型のポインターが指し示す値—整数 24680—を表示させる．
4. 整数型の変数の先頭アドレスを表示させる．
5. 整数型のポインターの先頭アドレスを表示させる．

［**練習 4**］ 以下のように動作するプログラムを作成せよ．

1. 倍精度実数型の変数を用意する．それに，実数 3.1415 を格納する．
2. 倍精度実数型のポインターを用意する．それに，先ほど用意した倍精度実数型の変数の先頭アドレスを格納する．
3. 倍精度実数型のポインターが指し示す値—実数 3.1415—を表示させる．
4. 倍精度実数型の変数の先頭アドレスを表示させる．
5. 倍精度実数型のポインターの先頭アドレスを表示させる．

5 課題

次の講義 (2 月 16 日) の AM8:45 までに，以下の課題をレポートとして提出すること．表紙等は，いつもの通り．表紙のタイトルは「コンピューターのメモリーとポインター」とすること．

［**問 1**］ (予) 教科書 p.270–292 を 2 回読み，重要と思うところに，赤い色の下線あるいは蛍光マーカで印を付けよ．レポートには「2 回読み，重要部分に線を引いた」と書け．

［**問 2**］ (復) 本日配布したプリントを 2 回読め．ただし，付録は興味のある者のみでよい．レポートには「2 回読んだ」と書け．そして，誤字脱字，日本語の文章のおかしなところ，間違いがあれば，レポートに記述せよ．

［**問 3**］ (復) メモリーのアドレスとは何か? 簡潔に説明せよ．

［**問 4**］ (復) ポインター変数とは何か? 簡潔に説明せよ．

［**問 5**］ (復) アドレス演算子 (&) と間接参照演算子 (*) の動作を説明せよ．

［**問 6**］ (復) リスト 3 プログラムの動作結果を示せ．

［**問 7**］ (復) 以下のように動作するプログラムを作成せよ．

1. 文字型の変数を用意する．それに，文字'A' を格納する．
2. 文字型のポインターを用意する．それに，先ほど用意した文字型の変数の先頭アドレスを格納する．
3. 文字型のポインターが指し示す値—文字'A'—を表示させる．
4. 文字型の変数の先頭アドレスを表示させる．
5. 文字型のポインターの先頭アドレスを表示させる．

リスト 3: 間接演算子と積の演算子が混在した式

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a=2, b=4, c=6;
6    int *ap, *bp, *cp;
7
8    ap=&a;
9    bp=&b;
10   cp=&c;
11
12   printf("c=%d\n",*ap**bp**cp);
13
14   return 0;
15 }
```

付録 A　付録

付録 A.1　型のサイズを調べる

型のサイズ (バイト数) は，`sizeof(型)` 関数を使えば，取得できる．

リスト 4: データ型によるバイト数調査プログラム

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6    printf("----- size ---------\n");
7    printf("\tchar\t%d\n", sizeof(char));
8    printf("\tint\t%d\n", sizeof(int));
9    printf("\tdouble\t%d\n", sizeof(double));
10
11   return 0;
12 }
```

実行結果

```
----- size ---------
        char    1
        int     4
        double  8
```

付録 A.2　変数のアドレスと内容を調べる

これは，無理矢理変数の中身をみるプログラムである．こんなもの，今は理解できなくてもよい．参考のために載せているだけだ．

リスト 5: データのアドレスと内容の調査プログラム

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6    double x=-7.696151733398438e-4;
7    int i=55;
8    char a='a';
9    unsigned char *p;
10
11   printf("--- char a ---------------\n");
12   printf("%p\t%02x\n", &a, a);
13
14   p=(unsigned char *)&i;
15
16   printf("--- int i ----------------\n");
17   printf("%p\t%02x\n", p,   *p);
18   printf("%p\t%02x\n", p+1, *(p+1));
19   printf("%p\t%02x\n", p+2, *(p+2));
```

```
20      printf("%p\t%02x\n", p+3, *(p+3));
21
22      p=(unsigned char *)&x;
23
24      printf("--- double x ---------------\n");
25      printf("%p\t%02x\n", p,   *p);
26      printf("%p\t%02x\n", p+1, *(p+1));
27      printf("%p\t%02x\n", p+2, *(p+2));
28      printf("%p\t%02x\n", p+3, *(p+3));
29      printf("%p\t%02x\n", p+4, *(p+4));
30      printf("%p\t%02x\n", p+5, *(p+5));
31      printf("%p\t%02x\n", p+6, *(p+6));
32      printf("%p\t%02x\n", p+7, *(p+7));
33
34      return 0;
35  }
```

実行結果

```
--- char a ---------------
0xbffff6ab      61
--- int i ----------------
0xbffff6ac      37
0xbffff6ad      00
0xbffff6ae      00
0xbffff6af      00
--- double x -------------
0xbffff6b0      00
0xbffff6b1      00
0xbffff6b2      00
0xbffff6b3      00
0xbffff6b4      00
0xbffff6b5      38
0xbffff6b6      49
0xbffff6b7      bf
```

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　基礎編　第 2 版. (株) オーム社, 2006.