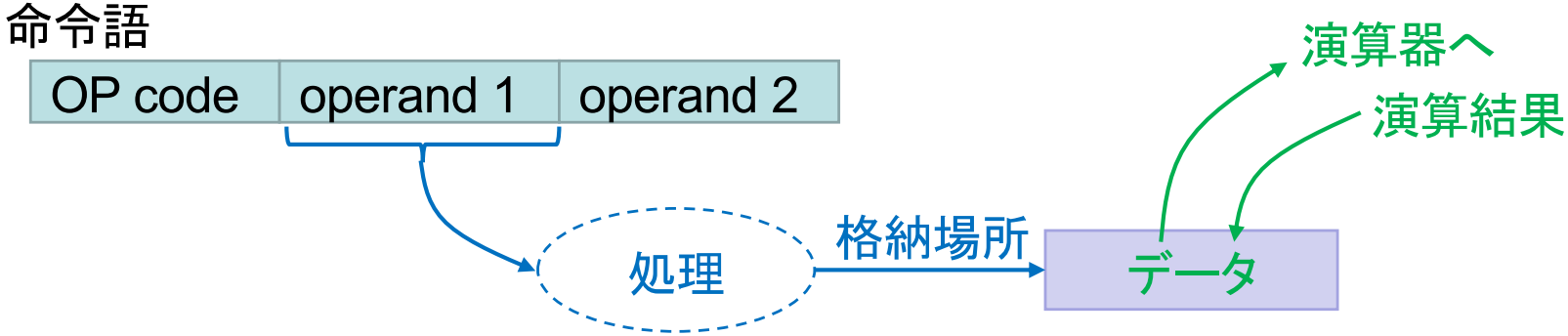
はじめに

- 今回は、アドレッシングモード（Addressing Mode）について講義します。

- 教科書の対応範囲は、以下の通りです。
  - 2.2.3項（p.43～52）

アドレッシングモードとは？

- データの格納場所を命令のオペランドフィールドで指定する方法、または、オペランドフィールドからデータの格納場所を求める方法
  - 日本語では「アドレス指定モード」や「アドレス修飾」「アドレス方式」などと訳す場合もあります。



- データの格納場所
  - レジスタ：　レジスタ番号でデータの格納場所を識別します。
  - メモリ：　アドレスでデータの格納場所を識別します。
    - アドレスで場所を指定できる範囲を「アドレス空間」「メモリ空間」と呼ぶことがあります。
    - 命令のオペランドフィールドから求めた最終的なメモリアドレスを実効アドレス（effective address）という。

絶対アドレスと相対アドレス

- メモリのアドレスをどのような考え方で指定するか？　を考えます。
  - 「メモリにはアドレスを付けている（アドレスが付いている）のだから、それをそのまま使いましょう」というのが絶対アドレス（absolute address）です。
    - つまり、メモリの2016番地は、絶対的に2016番地なのです。（当たり前すぎて何を言っているかわからないかもしれませんが。。。）
    - しかし、メモリ（主記憶）は容量が大きいので、その（絶対）アドレスを指定するには多くのビット数が必要になります。

- 「あるアドレスを基準として、そこからどれだけ離れているか、でデータの場所を指定しましょう」というのが相対アドレス（relative address）です。
  - 基準とするアドレスをベースアドレス（base address）と言います。
  - ベースアドレスからどれだけ（何番地分）離れているのかを偏位（displacement）やオフセット（offset）と言います。
  - このような考え方でアドレスを指定するのが相対アドレス（指定）であり、この場合、偏位（オフセット）を相対アドレスといいます。
    - 「相対アドレス」という言葉自体が考え方（指定方法）を指していたり、偏位を指していたりで、曖昧に使われるようです。

- 一般に、プログラムでは、離れた場所に分散しているデータにあちこちアクセスするということはなく、ある程度近い場所に格納されているデータにアクセスする性質があります。そこで共通のベースアドレスをもとに、そこからの相対アドレスで指定することにすると、相対アドレスは少ないビット数でよい、ということになります。ただし、実際にメモリ（装置）にアクセスするには絶対アドレスでなければならないので、ベースアドレスと相対アドレス（偏位、オフセット）とを加算して、絶対アドレスを求めなければなりません。
- ちなみに、「絶対アドレスは0番地をベースアドレスとした相対アドレスである」と解釈することができますが、そんな考え方をしても何のメリットもありません（つまり、無意味です）。

– 実効アドレスは、絶対アドレスとして求めます。

- 上で述べたように、実際にメモリ装置にアクセスするには、そのアドレス（つまり絶対アドレス）を指定しなければならないので。

- では、次ページから、個々のアドレッシングモードについて説明していきます。
  - コンピュータによっては、それらの中の一部のアドレッシングモードしか用いないものもあります。
  - ちなみに、皆さんのPCに搭載されているx86アーキテクチャのプロセッサでは、ここで説明する全てのアドレッシングモードを用意しています。

(1) 直接(direct)

- オペランドフィールドに、メモリのアドレス(絶対アドレス)を記載するモード。
  - つまり、このオペランドに書かれているアドレスが実効アドレス。
  - メモリの容量に応じて、オペランドに多くのビット数が必要。

- 教科書の図2.17(p.47)を参照
  - オペランドフィールドには102が2進数で記載されているので、メモリの102番地が実効アドレス。
  - ソースオペランドの場合は、102番地に格納されている値312(これも実際には2進数で格納されている)を読み出して演算に使用する。
  - デスティネーションオペランドの場合は、102番地に演算結果を書き込む。

- プログラミング言語との対応
  - 一般変数（C言語なら外部変数）にアクセスするのに利用できる。
  - 例えば、教科書の図2.17の例では、変数aがメモリの102番地に割り付けられていて、aの値が312である場合に対応する。

(2) 間接（indirect）

- オペランドフィールドに、アクセスしたいデータのアドレスを格納しているポインタのアドレス（絶対アドレス）を記載するモード。
- 教科書の図2.18（p.48）を参照
  - オペランドフィールドには850が2進数で記載されているので、メモリの850番地にアクセスしてその内容101を読み出す。この101が実効アドレスとなる。
  - ソースオペランドの場合は、101番地に格納されている値312を読み出して演算に使用する。
  - デスティネーションオペランドの場合は、101番地に演算結果を書き込む。

- 命令語をプログラム中に書き換えることは行わない。しかし、850番地の内容を書き換えることにより、命令語を変更することなく、アクセスするデータを動的に変更することができる。

実行時に行う場合を”動的”、実効前に行う（実効時には変更しない）場合を”静的”、という。

- プログラミング言語との対応
  - ポインタを介してデータにアクセスするのに利用できる。
    - 教科書の図2.18の例では、「`int a, *p=&a;`」とすると、850番地がポインタ変数`p`のアドレス（`&p`）、変数`p`の値が101、変数`a`のアドレス（`&a`）が101番地、変数`a`（つまり`*p`）の値が312、である場合に対応する。

(3) 即値（immediate）

- オペランドフィールドに、アクセスしたいデータの場所ではなく、データの値そのものを記載するモード。
  - 1、2、5、−1 など、よく使用する小さな（つまり表現に必要なビット数が少ない）定数を、メモリにアクセスすることなく演算に使用するのに用いる。
- 教科書の図2.19（p.48）を参照
  - オペランドフィールドに値312が2進数で記載されているので、これを演算に使用する。

(4) レジスタ直接

- オペランドフィールドに、レジスタの番号を記載するモード。
- 教科書の図2.20（p.49）を参照
  - オペランドフィールドには3が2進数で記載されている。
  - ソースオペランドの場合は、3番のレジスタに格納されている値312（これも実際には2進数で格納されている）を読み出して演算に使用する。
  - デスティネーションオペランドの場合は、3番のレジスタに演算結果を書き込む。

- プログラミング言語との対応
  - 一般変数（C言語なら外部変数）にアクセスするのに利用できる。
    - 直接（direct）のモードとは、変数がメモリに割り付けられているかレジスタに割り付けられているかの違い。
  - 例えば、教科書の図2.20の例では、変数aが3番のレジスタに割り付けられていて、aの値が312である場合に対応する。

(5) レジスタ間接

- オペランドフィールドに、アクセスしたいデータのアドレスを格納しているレジスタの番号を記載するモード。
- 教科書の図2.21（p.49）を参照
  - オペランドフィールドには4が2進数で記載されているので、4番のレジスタにアクセスしてその内容870を読み出す。この870が実効アドレスとなる。
  - ソースオペランドの場合は、メモリの870番地に格納されている値312を読み出して演算に使用する。
  - デスティネーションオペランドの場合は、メモリの870番地に演算結果を書き込む。

  - プログラム実行中に、他の命令で4番のレジスタの内容を書き換えることにより、アクセスする場所（アドレス）を動的に変更することができる。
- プログラミング言語との対応
  - ポインタを介してデータにアクセスするのに利用できる。
    - 間接（indirect）のモードとは、ポインタ変数がメモリに割り付けられているかレジスタに割り付けられているかの違い。
    - 教科書の図2.21の例では、「`int a, *p=&a;`」とすると、ポインタ変数`p`が4番のレジスタに割り付けられていて、その変数`p`の値が870、変数`a`のアドレス（`&a`）が870番地、変数`a`（つまり`*p`）の値が312、である場合に対応する。

(6) インデックス

- 以下の2つを1セットとしてオペランドフィールドに記載するモード。
  - ベースアドレス
  - オフセットを格納するレジスタの番号
    - この役割を果たすレジスタをインデックスレジスタと呼ぶ。
- 教科書の図2.22（p.50）を参照
  - オペランドフィールドには125と3が、それぞれ2進数で記載されている。125がベースアドレスで、インデックスレジスタとして用いるのが3番のレジスタである。3番のレジスタの内容−25を125に加えて得られる100が実効アドレスとなる。

- プログラム実行中に、インデックスレジスタの内容を書き換えることにより、アクセスする場所（アドレス）を動的に変更することができる。

- プログラミング言語との対応
  - 1次元配列の要素にアクセスするのに利用する。
    - 例えば、以下のプログラムの場合、

```
char  a[100];
for( int i=0; i<100; i++ )
{
     … a[i] …
}
```

    - 配列a[]の先頭アドレス（つまり、a[0]のアドレス）を125番地とすると、a[0]、a[1]、a[2]…と順に並んでいるので、a[i]のアドレスは（125+i）番地。
    - 教科書の図2.21の例で変数iを3番のレジスタに割り付ければ、配列の要素a[i]にアクセスできる。

- しかし、配列がchar型ではなくてint型だったとすると？
  - int型のデータのサイズは4バイト。
  - 配列a[]の先頭アドレス（つまり、a[0]のアドレス）を100番地とすると、a[i]のアドレスは（100+i×4）番地。
  - これではインデックス方式（モード）は役に立たない。
- そこで、考え出されたのがScaled Index方式（モード）
  - ×2、×4、×8、×16などの2^nの倍率をオペランドに指定できるようにして、（ベースアドレス）＋（インデックスレジスタの値）×（倍率）で実効アドレスを求める。
- しかし、それでも用途は限られる。
  - 構造体（サイズはプログラマの定義次第で任意）の配列には歯が立たない。

(7) ベース

- 以下の2つを1セットとしてオペランドフィールドに記載するモード。
  - ベースアドレスを格納するレジスタの番号
    - この役割を果たすレジスタをベースレジスタと呼ぶ。
  - オフセット
- 教科書の図2.24（p.51）を参照
  - オペランドフィールドには3と20が、それぞれ2進数で記載されている。ベースレジスタとして用いるのが3番のレジスタで、20がオフセットである。3番のレジスタの内容800に20を加えて得られる820が実効アドレスとなる。

- プログラム実行中に、ベースレジスタの内容を書き換えることにより、アクセスする場所（アドレス）を動的に変更することができる。

- プログラミング言語との対応
  - 構造体へのポインタを用いてその構造体メンバにアクセスするのに利用できる。
    - 例えば、以下の構造体定義の場合、

```
struct sample {
    int a, b, c;
} *p;
```

      - この構造体の先頭アドレスを*x*番地とすると、`int`型は4バイトなので、メンバ`a`、`b`、`c`のアドレスはそれぞれ*x*、*x*+4、*x*+8番地である。例えば`p->c`をオペランドに指定したければ、変数`p`の値を入れたレジスタの番号とオフセットの8を指定すればよい。

- （参考）リロケーション（relocation）
  - 複数のデータなどを固めて1つのブロックとする。
  - ブロック内の個々のデータは、すべて、ブロックの先頭アドレスからの相対アドレスで指定する。
  - このブロックがメモリ上のどこに配置されても、先頭アドレス＋相対アドレスで全てのデータにアクセスできる。
    - つまり、ブロックの先頭アドレスをベースレジスタに記憶して、ベース方式（モード）でアドレッシングする。
  - 以上のようなブロックを再配置可能（relocatable）なブロックという。
  - ライブラリなどは再配置可能な形式で構成されている。
    - printf()などの関数はコンパイルされた状態でライブラリとして用意されている。どんなアプリケーションプログラムとリンクされるかわからない（したがって、そのプログラムのサイズもわからない）ので、ライブラリだけでアドレスを決めてしまうことができない。そこで、上記のように再配置可能な形式で準備しておき、実際にアプリケーションプログラムとリンクするときに、メモリ上のどこに配置されてもよいようにしている。

ベース vs インデックス

- ベースアドレスとオフセットを加えて実効アドレスを求める点で、両者は似ている。
- しかし、どちらをオペランドフィールドに直接書き込むかで以下の違いが生じる。
  - ベースアドレスはメモリアドレスをフルに指定する必要があるので、インデックス方式（モード）では、命令語長が長くなる。
  - オフセットは必ずしもメモリアドレスをフルに指定する必要がない（ベースアドレスの近傍にしかアクセスしないという前提）ので、ベース方式（モード）では、命令語長が長くなるのを防ぐことができる。
    - 上記の前提は、逆に、アクセスできる範囲に制限を設けることになるが、その前提が正しい限り（そして、実際は正しい）、その制限は問題にならない。

(8) ベースインデックス

- ベース方式（モード）とインデックス方式（モード）の両方を組み合わせたモード。
- 以下の2つを1セットとしてオペランドフィールドに記載するモード。
  - ベースアドレスを格納するレジスタ（ベースレジスタ）の番号
  - オフセットを格納するレジスタ（インデックスレジスタ）の番号
  - 場合によっては、さらに以下のものを加えたモードを用いる命令セットもある。
    - 追加のオフセット
    - インデキシング用のスケール（倍率）

- 以下は、前回の講義で使用した例題プログラムです。
  - 1行目のadd命令の第2ソースオペランドのアドレッシングモードはベースインデックス（倍率付き）です。
  - 実効アドレスはebx+edx×4で求めます。
    - つまり、ebxをベースレジスタとして、また、edxをインデックスレジスタとしてそれぞれ使用し、倍率に4を指定しています。

```
命令アドレス    命令語          アセンブリ表記
-----------------------------------------------
8048580:    03 04 93    add   (%ebx,%edx,4),%eax
8048583:    42          inc   %edx
8048584:    39 ca       cmp   %ecx,%edx
8048586:    7c f8       jl    8048580
```

(9) プログラムカウンタ(PC)相対

- プログラムカウンタ(Program Counter; PC)の内容をベースアドレスとして、オペランドフィールドには、そこからのオフセットを記載するモード。
- プログラムカウンタとは？
  - 次に実行する命令のアドレスを格納するレジスタ
    - 命令の実行中に、次の命令アドレスを指すようにPCを更新するので、更新前のPCは、まだ、“現在実行中”の命令アドレスを指していることになる。

- 教科書の図2.26（p.52）を参照
  - オペランドフィールドには42が2進数で記載されているので、PCに格納されている命令アドレス300に42を加えて、432が実効アドレスとなる。
- 多くの場合は命令アドレスをオペランドで指定する場合に使われますが、データアドレスを指定するのに使う命令セットもあります。
- PCが現命令アドレスを指すのか、それとも次命令アドレスを指すのかについては、命令セットによって定義が異なります。

- 以下は、前回の講義で使用した例題プログラムです。
  - jl命令のオペランドには条件が成立した場合の分岐先アドレスを指定しますが、命令語には1バイトで f8 としか書かれていません。
  - オペランドが1バイトなのは、アドレッシングモードとしてPC相対を用いているからです。
    - アドレッシングモードが“直接”であれば、4バイトの長さで08048580と記述しなければならないところです。
  - jl命令の命令アドレスは8048586番地で、その次の命令アドレスは8048588番地です。一方、f8 を1バイトの符号付き整数とみなすと、10進数で −8 を表します。したがって、8048588に−8 を加えて、分岐先のアドレスが8048580番地と求まります。

```
命令アドレス    命令語        アセンブリ表記
-----------------------------------------
8048580:    03 04 93    add   (%ebx,%edx,4),%eax
8048583:    42          inc   %edx
8048584:    39 ca       cmp   %ecx,%edx
8048586:    7c f8       jl    8048580
```