アセンブリ言語

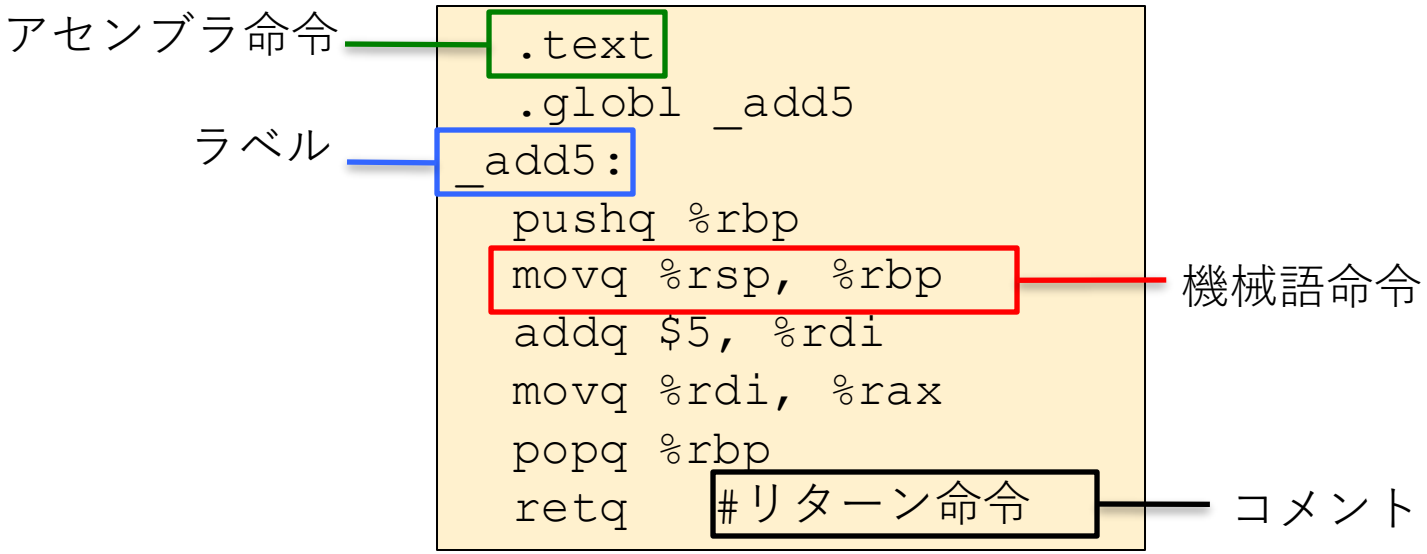
イントロダクション（2）

情報工学系
権藤克彦

機械語命令，アセンブラ命令，ラベル，コメント

- 機械語命令
- アセンブラ命令
- ラベル
- コメント

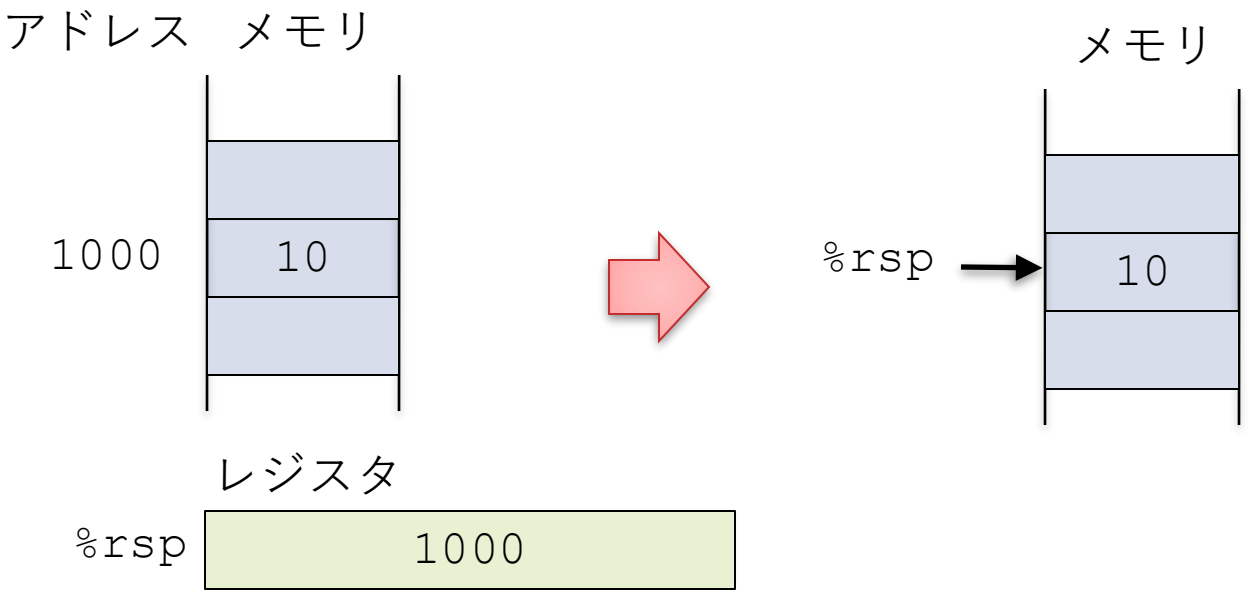
アセンブリコードは
この4つで書く．
（基本的には行単位）



インデントは見やすさのためだけ

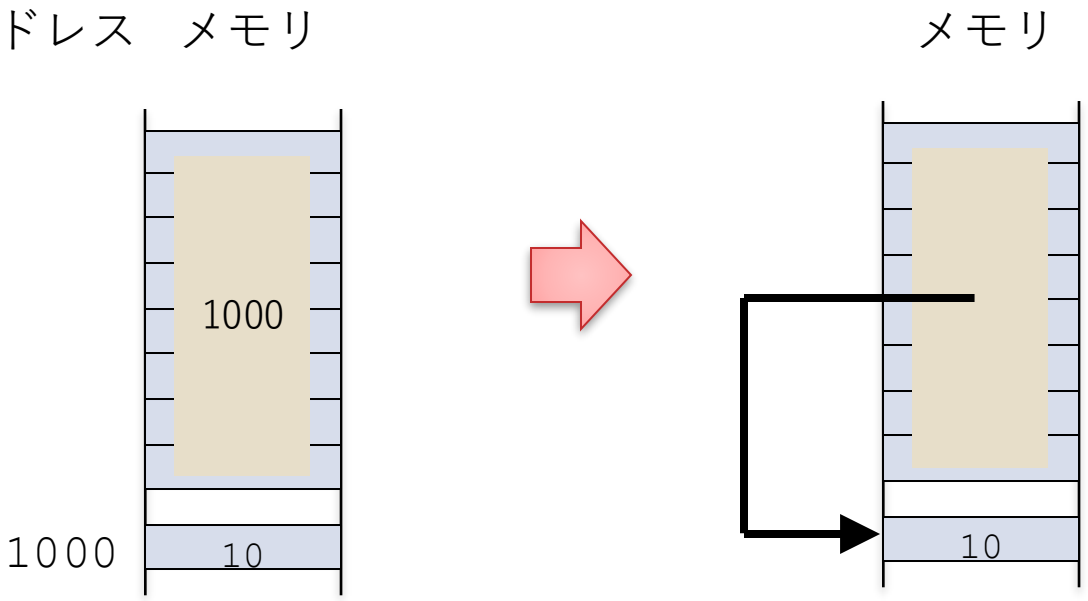
ポインタの図表現（1）

- レジスタ%rspはメモリのアドレス1000番地を値として保持.
  - これは「%rspは1000番地を指している」ことを示す.
  - これを矢印で右図のように図表現する.



ポインタの図表現（2）

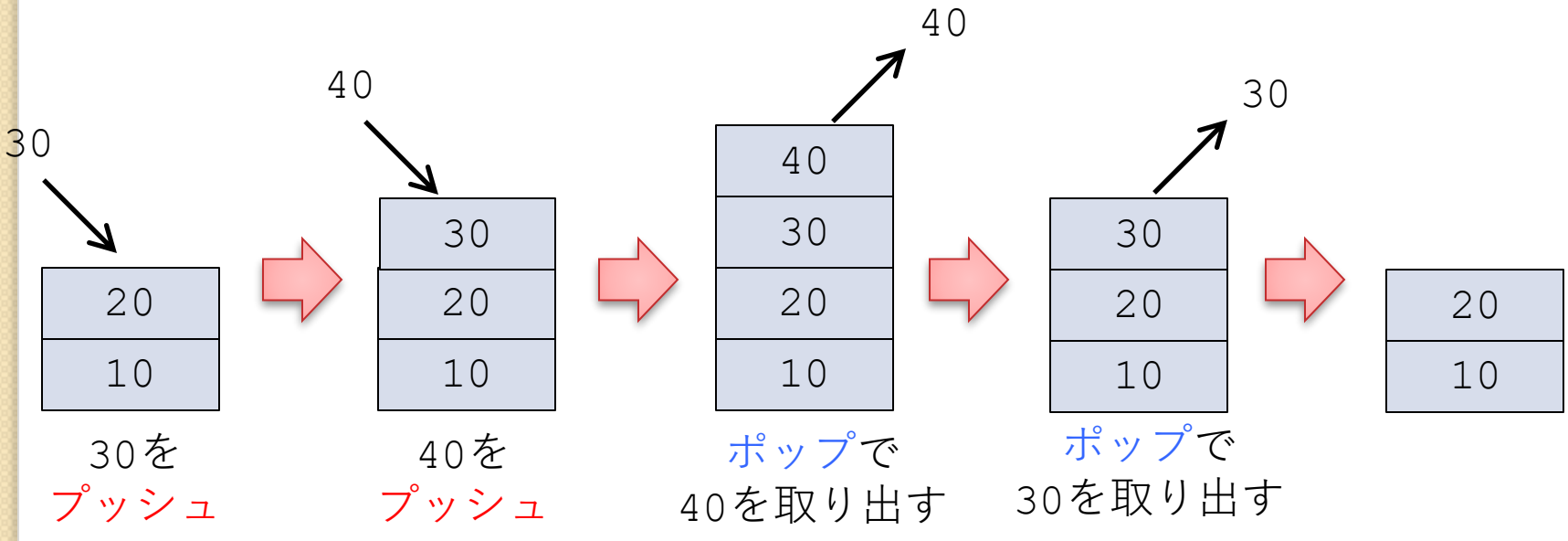
- メモリがメモリを指す場合も同様に矢印で図表現.
  - x86-64のアドレス長は64ビット（＝8バイト）



複数バイトのデータは複数のアドレスにまたがる
→ 先頭アドレス（一番小さいアドレス）を代表アドレスに

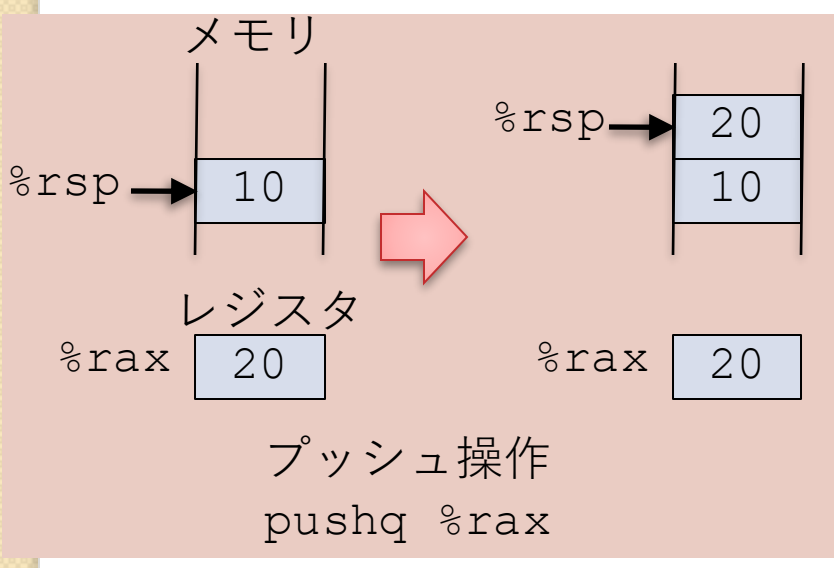
スタック (stack)（１）

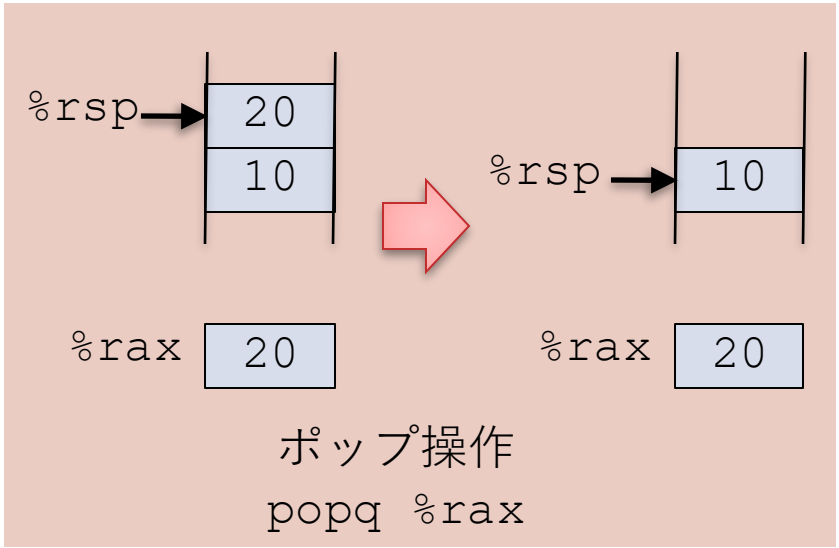
- LIFO (last-in first-out) のデータ構造.
  - 後に入れたデータが，先に取り出される．cf. キュー(queue)
- プッシュ操作とポップ操作がある.
  - プッシュ(push)＝スタックトップにデータを積む.
  - ポップ(pop)＝スタックトップからデータを取り出す.



スタック (stack)（２）

- 言語処理系では関数（手続き）呼び出しの実装に利用
- スタックフレーム(stack frame)
  - 関数呼び出し1回分のデータ.
  - 局所変数，引数，戻り番地，返り値などを含む.
- %rspはスタックトップを指す（ようにする）.
  - %rsp は「スタックポインタ」と呼ばれる.





add5.s 早わかり

add5.c

```
long add5 (long n)
{ return n+5; }
```

add5.s

```
   .text
   .globl _add5
_add5:
   pushq %rbp
   movq %rsp, %rbp
   addq $5, %rdi
   movq %rdi, %rax
   popq %rbp
   retq
```

以降を.textセクションに出力.
このラベルをグローバル（大域）にせよ.
ラベル（アドレスはアセンブラが自動計算）.

```
%rbpの値をスタック上に退避.
%rbp = %rsp
%rdi += 5
%rax = %rdi

popq %rip  （リターン）
```

新しいスタックフレームを作成

スタックフレームを破棄.

この命令は実在しない

手続き呼び出し規約（一部）
・第1引数は %rdi で渡す
・返り値は %rax で返す

- ドット記号（.）で始まるのはアセンブラ命令.
- ドル記号（$）は定数（即値）.
- パーセント記号（%）はレジスタ.
- コロン記号（:）で終わるのはラベル.

コメント（1）

- プログラマのメモ書き．アセンブラは単に無視する．
- 2種類のコメントを使える．
  - 行コメント：
    - x86-64の場合はシャープ記号（＃）から行末までがコメント．
      ちなみにSPARCではビックリ（！），H8ではセミコロン（；）．

```
#    これは行コメントです．
```

  - ブロックコメント：
    - C言語のブロックコメントと同じ．/* から */ までがコメント．
      ネスト（入れ子）禁止．

```
/*    これはブロックコメントです．
        複数行でもOKです．        */
```

コメント（2）

- 拡張子を.S（大文字）にする→C前処理命令を使える.
  - #if を使って，入れ子可能なコメントを書ける.

```
#if 0
これはC前処理命令を使った
コメントです.
#endif
```

#define や #include も使用可.

機械語命令 (machine instruction)

- CPUが実行する命令．マシン語ともいう．
  - cf. アセンブラ命令はアセンブラが実行する．
  - CPUが直接実行できるのは（2進数の）機械語だけ．

- 例：`movq %rsp, %rbp`
  - 「%rspレジスタ中の値を%rbpレジスタにコピーせよ」
  - ただし，これは記号表現（ニモニック）．このままではCPUは実行できない．
  - この命令の2進数表現は 01001000 10001001 11100101
  - アセンブラがニモニックを2進数（バイナリ）に変換する．

ニモニック (mnemonic)

- 2進数の機械語命令の記号表現.
  - 例：`movq %rsp, %rbp`
- 英語の省略．人間が覚えやすいように.
  - 例：`movq = move quad`
- 機械語命令とニモニックはほぼ一対一に対応.

mnemonic
- 形：記憶を助ける，記憶術の
- 名：記憶を助ける工夫（公式・覚え歌など）

x86-64では
word = 2バイト
double word (long) = 4バイト
quad word = 8バイト

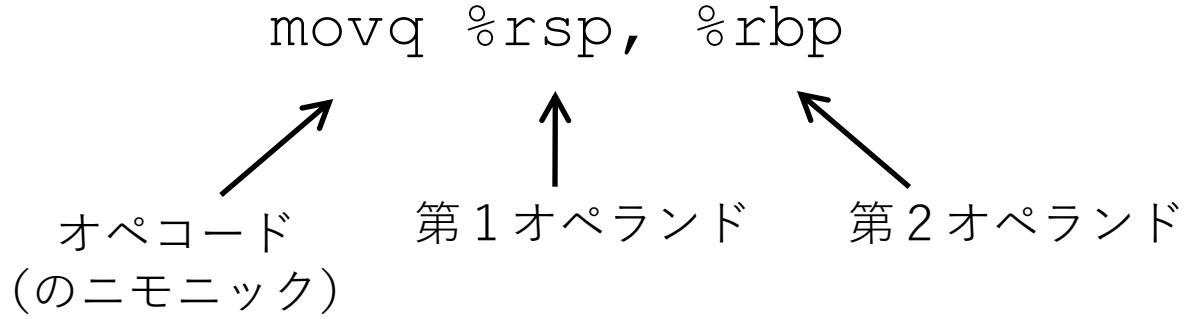
quadruple （4倍の）

ワード (word)の本来の意味は
「そのCPUが自然に扱う整数サイズ」
もともと 8086 が16ビットCPUだった名残

LP64メモリモデルでは
C言語のlong型は8バイト長

オペコードとオペランド

- オペコード(opcode)
  - 機械語命令のうち，処理の内容の部分.
    - 例：movq データのコピー
  - operation code の略.
- オペランド (operand)
  - 機械語命令のうち，処理の対象の部分.
    - 例：%rsp レジスタ
  - 便宜上，左から「第１オペランド」「第２オペランド」と呼ぶ.



AT&T形式とIntel形式

- x86-64用アセンブラは大きく2種類.
  - GNUアセンブラ（のデフォルト）はAT&T形式を使用.
  - 他のアセンブラ（NASM, MASM, TASMなど）やIntelのドキュメントではIntel形式を使用.

AT&T形式

```
pushq %rbp
movq  %rsp, %rbp
subq  $0x8, %rsp
```

代入は左から右

Intel形式

```
push  rbp
mov   rbp, rsp
sub   rsp, 0x8
```

代入は右から左

```
% objdump -d -M att -M suffix add5.o
```

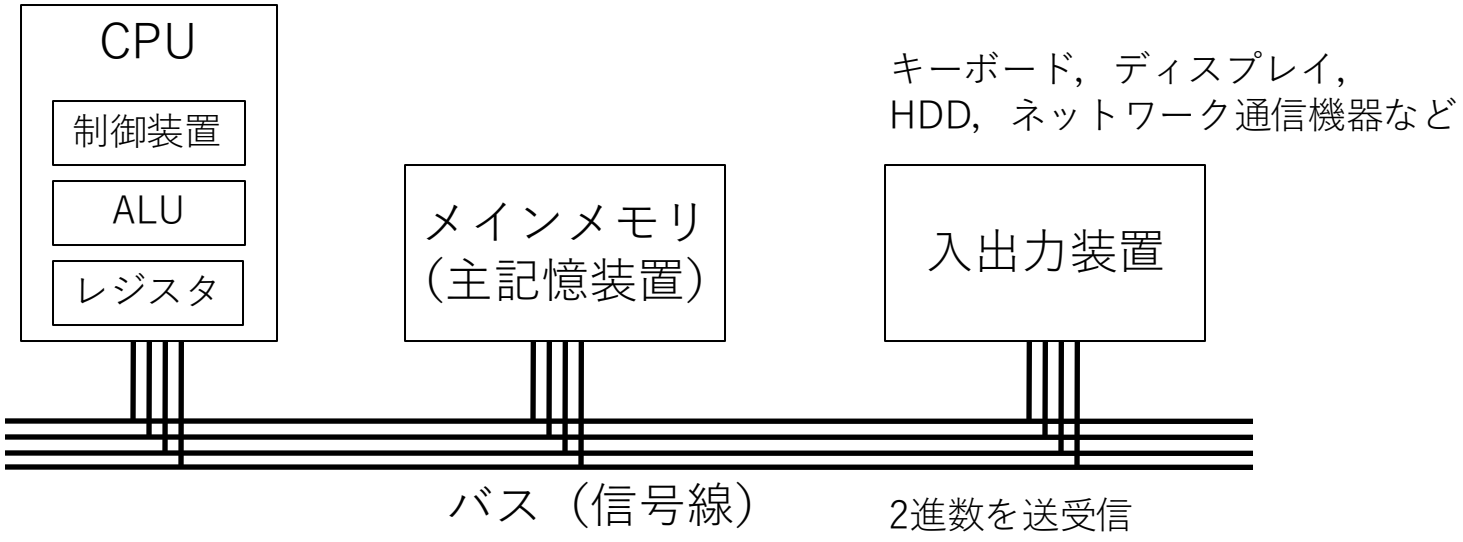
```
% objdump -d -M intel add5.o
```

機械語命令の主な種類

- 演算命令＝データを計算
  - add, sub, imul, idiv, neg, cmp, cltd, movsbq, movzbq
- 制御命令＝実行順序の変更
  - jmp, j*cc*（条件付きジャンプ）, call, ret
- 転送命令＝データのコピー
  - mov, push, pop, lea, set*c*
- その他
  - 特権命令，入出力命令（in, out）

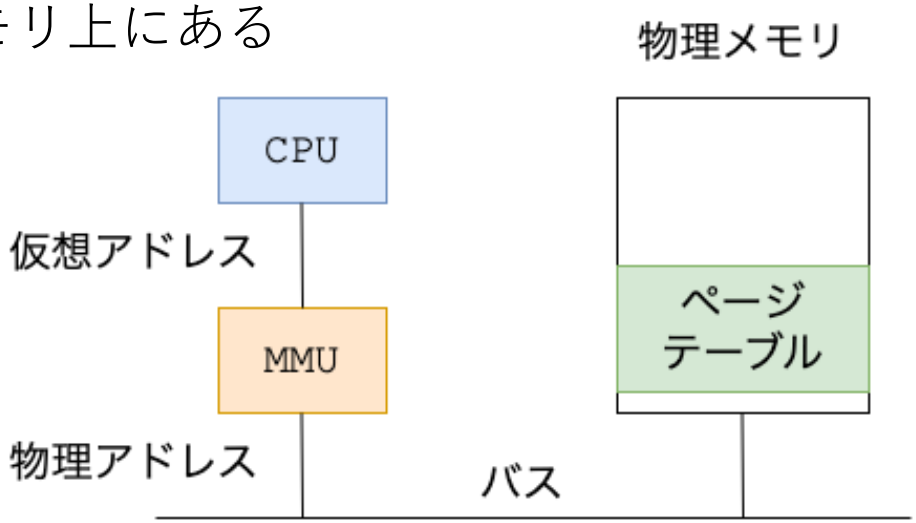
コンピュータの基本構造

- CPU（中央処理装置，プロセッサ）
  - 制御装置，ALU（演算装置），レジスタから構成.
- メインメモリ（主記憶）　以後，メモリと略す.
- 入出力装置
- バス(bus)



補足：CPUとMMU

- CPUとバスの間には MMU がある
  - memory management unit
- CPUが扱うアドレスはすべて仮想アドレス
  - 仮想メモリ機能が有効な場合は
- MMUが高速に仮想アドレス↔物理アドレスを変換
- ページテーブル＝仮想・物理アドレス対応表
  - 物理メモリ上にある

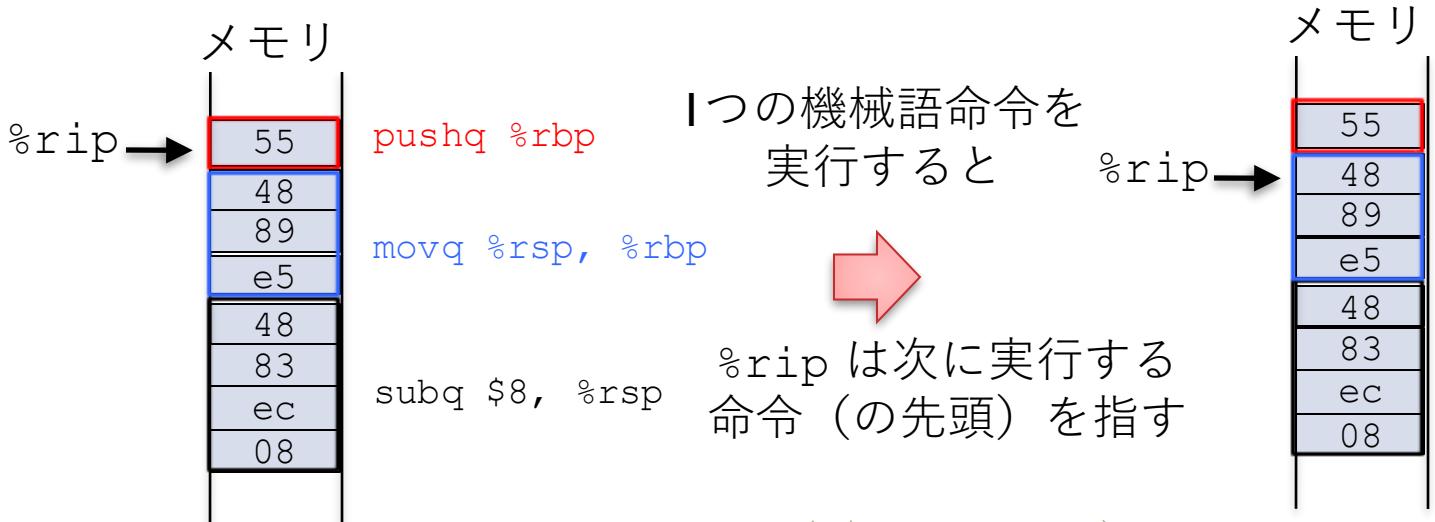


CPUの基本構造

- 制御装置
  - フェッチ実行サイクルをひたすら繰り返す.
- ALU
  - 四則演算や論理演算などを計算（例：addq）
- レジスタ
  - 高速で小容量・固定長のメモリ.
  - 専用レジスタ＝特定の役割を持つレジスタ.
    - 例：プログラムカウンタ（%rip）.
  - 汎用レジスタ＝様々な用途に使えるレジスタ.
    - 例：%rax, %rbx

プログラムカウンタ

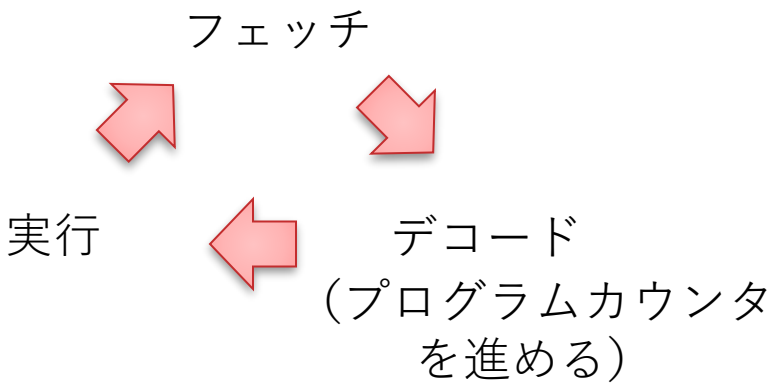
- 「次に実行する機械語命令を格納するメモリのアドレス」を保持.
- x86-64では %rip レジスタ.
  - movq命令などは，%ripレジスタにアクセス不可.
    - movq %rip, %rax とは書けない．movq 0(%rip), %rax とは書ける.
  - jmp命令やcall命令などで，間接的に%rip にアクセス.
    - %rip を書き換える＝実行をジャンプする.



簡略化してます.
特に割り込み処理.

フェッチ実行サイクル (fetch-execute cycle)

- CPUは次の動作をひたすら繰り返す.
  - フェッチ(fetch)
    - プログラムカウンタが指す機械語命令をメモリからCPUに読み込む.
  - デコード(decode)
    - 読み込んだ命令を解析して，実行の準備をする.
    - 次の機械語命令を指すようにプログラムカウンタの値を増やす.
  - 実行 (execute)
    - 読み込んだ機械語命令を実行する.



アセンブラ命令 (assembler directive)

- アセンブラ命令（例：`.text`）はアセンブラが実行.
  - cf. 機械語命令（例：`movq`）はCPUが実行.
  - CPUが実行しないので，疑似命令とも呼ばれる.
- すべてドット記号（.）で始まる.
  - GNUアセンブラの場合.

pseudo instruction,
pseudo opcode

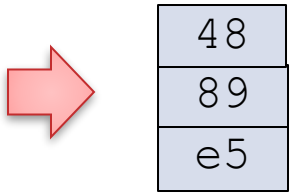
アセンブラ命令と機械語命令の違い

	例	何が実行	いつ実行	バイナリファイル中に
アセンブラ命令	`.text`	アセンブラ	アセンブル時	ない
機械語命令	`movq %rsp, %rbp`	CPU	実行時	ある

アセンブラの主な仕事（1）

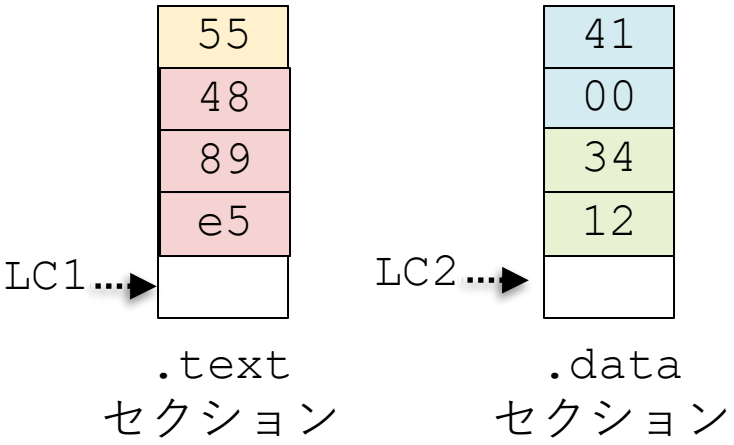
- 機械語命令のニモニックを2進数表現に変換.
  - アセンブラにとって機械語命令は処理対象のデータ.

```
movq %rsp, %rbp
```



- 2進数データをセクション毎に順番に出力.
  - ロケーションカウンタ(LC)で，出力バイト数を管理.

```
.text
pushq %rax
.data
.ascii "A\0"
.text
movq %rsp, %rbp
.data
.word  0x1234
```



0x10000FB8 + 0x48 = 0x100001000

%ripからの
相対アドレス

アセンブラの主な仕事（2）

- 記号表を作り，ラベルをアドレスに変換.
  - 例：ラベル _x をアドレス 0x00002014 に変換.

```
.text
movq %rax, _x
```

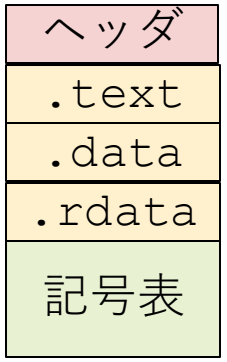
```
.data
.globl _x
_x:
.long 999
```

```
%objdump -d a.out
__text:
100000fb1:48 89 05 48 00 00 00
          movq %rax, 72(%rip)
100000fb8:
```

```
% nm a.out
0000000100000fb1 T _main
0000000100001000 D _x
```

記号表の内容

- 変換したデータをバイナリ形式のファイルとして出力.
  - macOS の場合は Mach-O形式.
  - ヘッダと複数のセクションから構成.



それぞれがセクション

アセンブラ命令の主な種類

- セクション指定

```
.text
```

以降を.textセクションに出力せよ.

- データ配置

```
.long 0x12345678
```

4バイトの整数値0x12345678の2進数表現を出力せよ.

- 出力アドレス調整

```
.align 4
```

4バイト境界にアラインメント調整せよ.
（4の倍数になるようにロケーションカウンタの値を増やせ.）

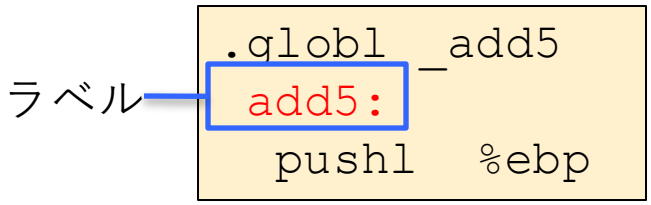
- シンボル情報

```
.globl _main
```

シンボル_mainをグローバルにせよ.
（記号表のエントリにフラグを立てる）

- その他

ラベル



- ラベルは機械語命令やアセンブラ命令の前に書ける.
  - たいてい，ラベルだけの行を書く.
- アセンブラが自動的にラベルをアドレスに変換する.

- 識別子（変数名，関数名）やジャンプ先を表すのにラベルを使う.
- オペランドにラベルを書いて良い.
  - アドレスが書ける場所なら.

```
movq %rax, _x(%rip)
```

インラインアセンブラ

- インラインアセンブラ (inline assembler)
  - 高級言語中にアセンブリコードの記述を可能にする.
  - 記述したものをインラインアセンブリコードという.
- アセンブリコードの記述量を減らせる.
  - アセンブリコードの生産性・保守性・移植性は低い.
- GCCではasm構文で記述する.

```
int main (void)
{
    asm ("nop");
}
```

nop命令を埋め込む単純な例

```
#include <stdio.h>
int main (void)
{
    void *addr;
    asm ("movq %%rsp, %0": "=m"(addr));
    printf ("rsp = %p\n", addr);
}
```

スタックポインタ(%rsp)の値を変数addrに格納する例