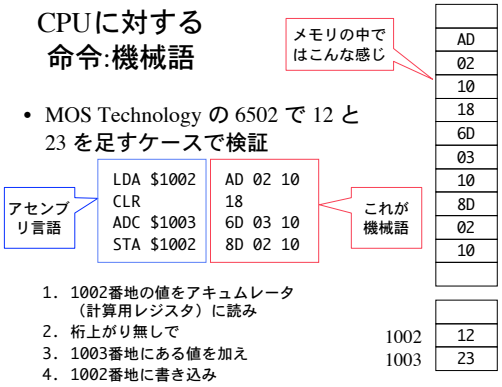
先週の復習: CPU が働く仕組み

- コンピュータの構造
  - 制御装置＋演算装置＋レジスタ
  - 制御装置がなければ電卓と同様
  - 制御装置＝自動的に命令どおり演算を実行する
- コンピュータの構造
  - CPU＋記憶装置（メモリやディスク）＋入出力装置＋etc..
- pp.100-101.

CPUに対する命令:機械語

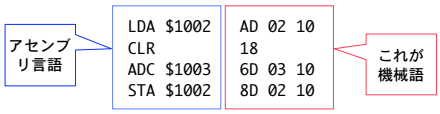


- MOS Technology の 6502 で 12 と 23 を足すケースで検証

1. 1002番地の値をアキュムレータ（計算用レジスタ）に読み
2. 桁上がり無しで
3. 1003番地にある値を加え
4. 1002番地に書き込み

アセンブリ言語

- 機械語を読みやすい形式で表現したもの
  - 6502の場合は 1 バイト 1 ニモニックで分かりやすいが、pp.118 のようにビット単位で解釈するものなども多いため、このように表現するだけでかなり読みやすくなる。
  - 実際には複雑な機能もあるがここでは紹介しない



マシン語とアセンブリ言語

- マシン語
  - ハードウェアにべったり依存
  - CPU設計時に言語仕様が決定
- アセンブリ言語
  - マシン語とほぼ一対一対応
- 実際にプログラミングする際には、どのCPU上で実行されるか意識してプログラミングする必要あり

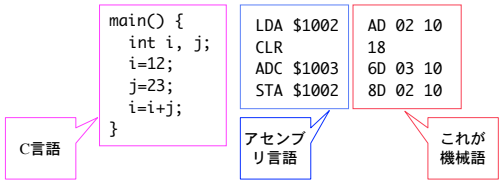
高水準言語

- 要求
  - ハードウェアに依存しないプログラミング
  - 長持ちして欲しい
  - いろんなシステムで動いて欲しい
- 結果
  - 高水準言語で書いて、低水準言語（または機械語）に変換すれば良い (pp.120)

そうでないシステム（組み込み系など）もある。そこでは高水準言語が不要な場合も少なくない。

高水準言語の例

- C言語の場合
  - メモリの構造やレジスタの名前を意識しないプログラムが書けている
  - 抽象度が高くなっている、と表現する



抽象度

- 二つの整数を加算する場合
- 機械語
  - 一時使用するレジスタ番号、メモリアドレスを指定
- アセンブリ言語
  - レジスタ番号などはまだ残る
- 高水準言語
  - ハードウェアの中身から独立している
  - 抽象度が高まった、移植性が高まった

47 F0 E0 57
F0 E4 E7 F0
E8

LD G1, F0E0
ADD G1, F0E4
ST G1, F0E8

a=b+c

pp.124~

自然言語と人工言語

- 自然言語
  - 人間が日常生活で用いている言語
  - 日本語、英語、etc..
- 人工言語
  - 人間が意図的に作り出した言語
  - エスペラント (1887年 L.L. Zamenhof , ポーランド)
  - 全てのプログラミング言語
- プログラミング言語の特徴
  - 決定的な動作のための明確な手順指示書
  - 一点一字の間違いも許されない
  - 限定的な語彙（C言語では予約語は 32）
  - 簡単な文法

具体例：アセンブリ言語

- 低水準プログラミング言語
  - 自然言語に近いほど「高水準」と考える
  - 歴史的には低水準から高水準へと進化
- 特徴
  - CPU 依存（移植性がない）CPU の機能を直接指定
  - 機械語とほぼ一対一
  - ニーモニックコード (mnemonic : 記憶を助ける)
  - 可読性が悪く、保守性が悪い
  - 作成が難しく、生産性が低い
  - 機械語に変換するソフトをアセンブラと呼ぶ

具体例：C

- 特徴
  - (アセンブラよりは)高水準な言語
  - CPU 依存性なし (CPU 命令は直接指定できない)
  - メーカー、OS が変わっても再利用できる
  - 移植性が高い
  - ニーモニックより更に抽象度が高く、それよりは自然言語に近い（プログラマにやさしい）語彙と文法
  - コンパイラ(Compiler)と呼ばれるソフトを利用してアセンブリ言語または機械語に変換
  - 可読性が高く、保守しやすい
  - プログラミングが容易で生産性が高い

抽象度

- 二つの整数を加算する場合
- 機械語
  - 一時使用するレジスタ番号、メモリアドレスを指定
- アセンブリ言語
  - レジスタ番号などはまだ残る
- 高水準言語
  - ハードウェアの中身から独立している
  - 抽象度が高まった、移植性が高まった

47 F0 E0 57
F0 E4 E7 F0
E8

LD G1, F0E0
ADD G1, F0E4
ST G1, F0E8

a=b+c