インフォマティクス1
第二回
- 計算機の仕組み２ -

名古屋大学 大学院情報学研究科
情報システム学専攻
本田　晋也

最終更新：2018年6月22日

アジェンダ

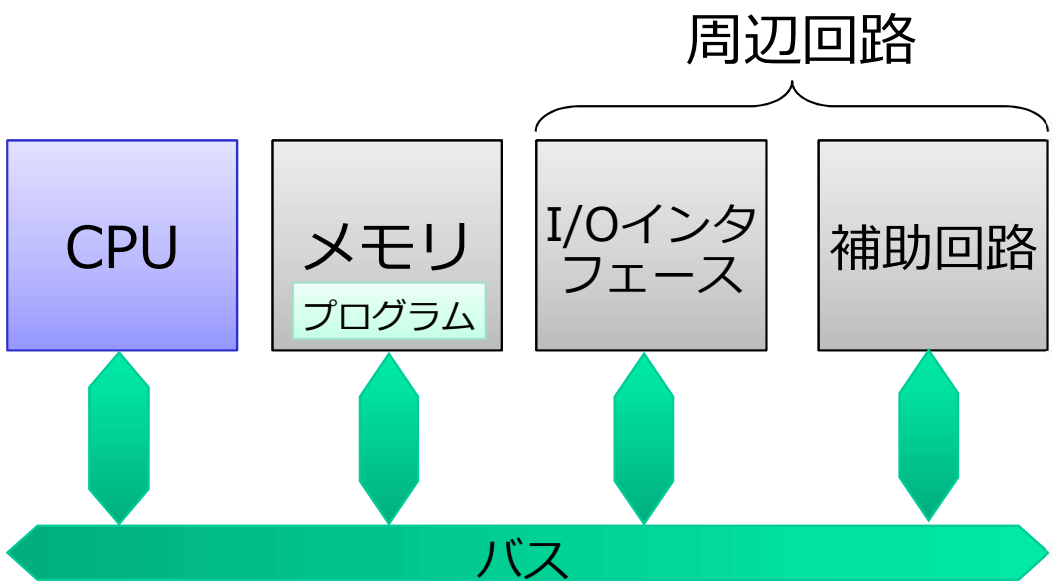
計算機の仕組みとして，
コンピュータの構成，プログラム言語，
ソフトウェアプラットホームについて説明

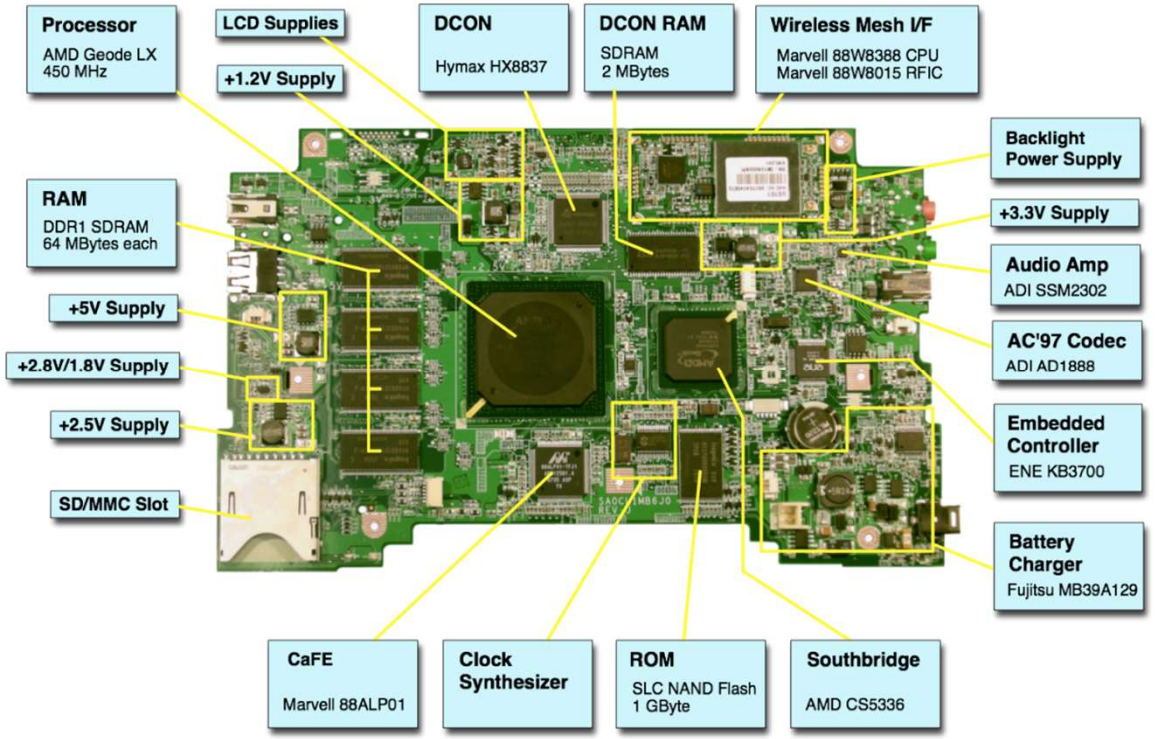
- コンピュータの構成
  - プロセッサ
  - 周辺回路
- プログラム言語
- ソフトウェアプラットホーム

コンピュータの構成

コンピュータの構成

- プロセッサ（CPU）を中心に，メモリやI/Oインタフェースなどがバスにより接続されている
- メモリにはプログラムが格納されており，プロセッサはプログラムを実行するによって，様々な処理を行う

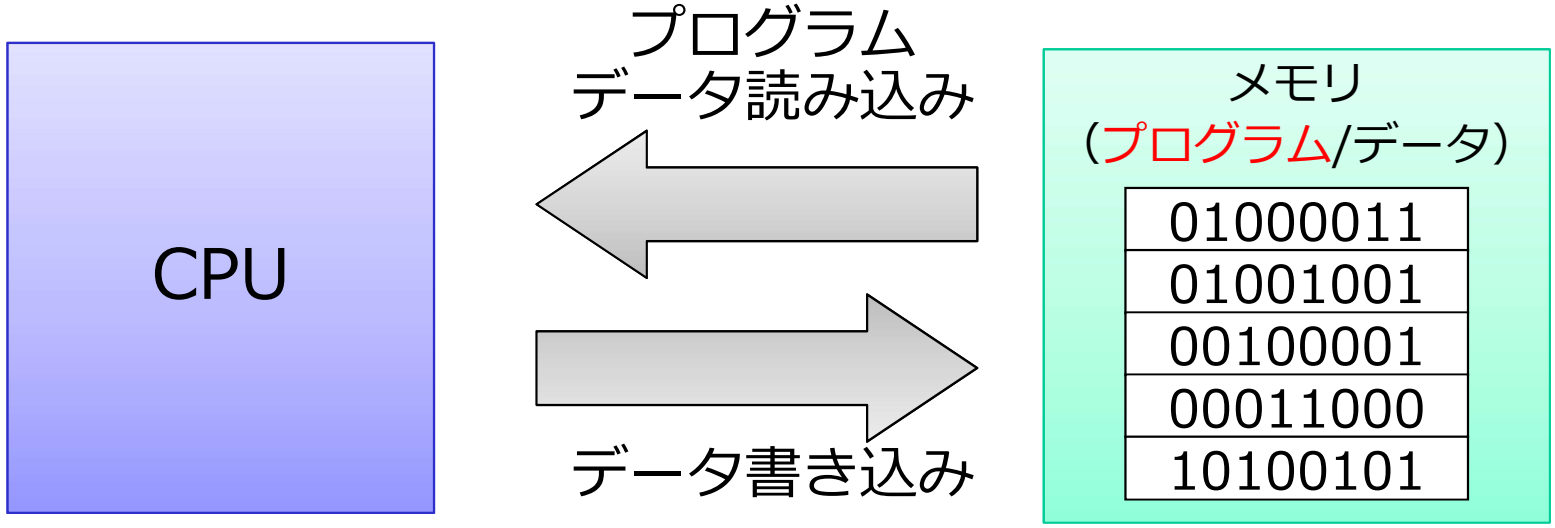




組込みシステムの基板

コンピュータの動かし方

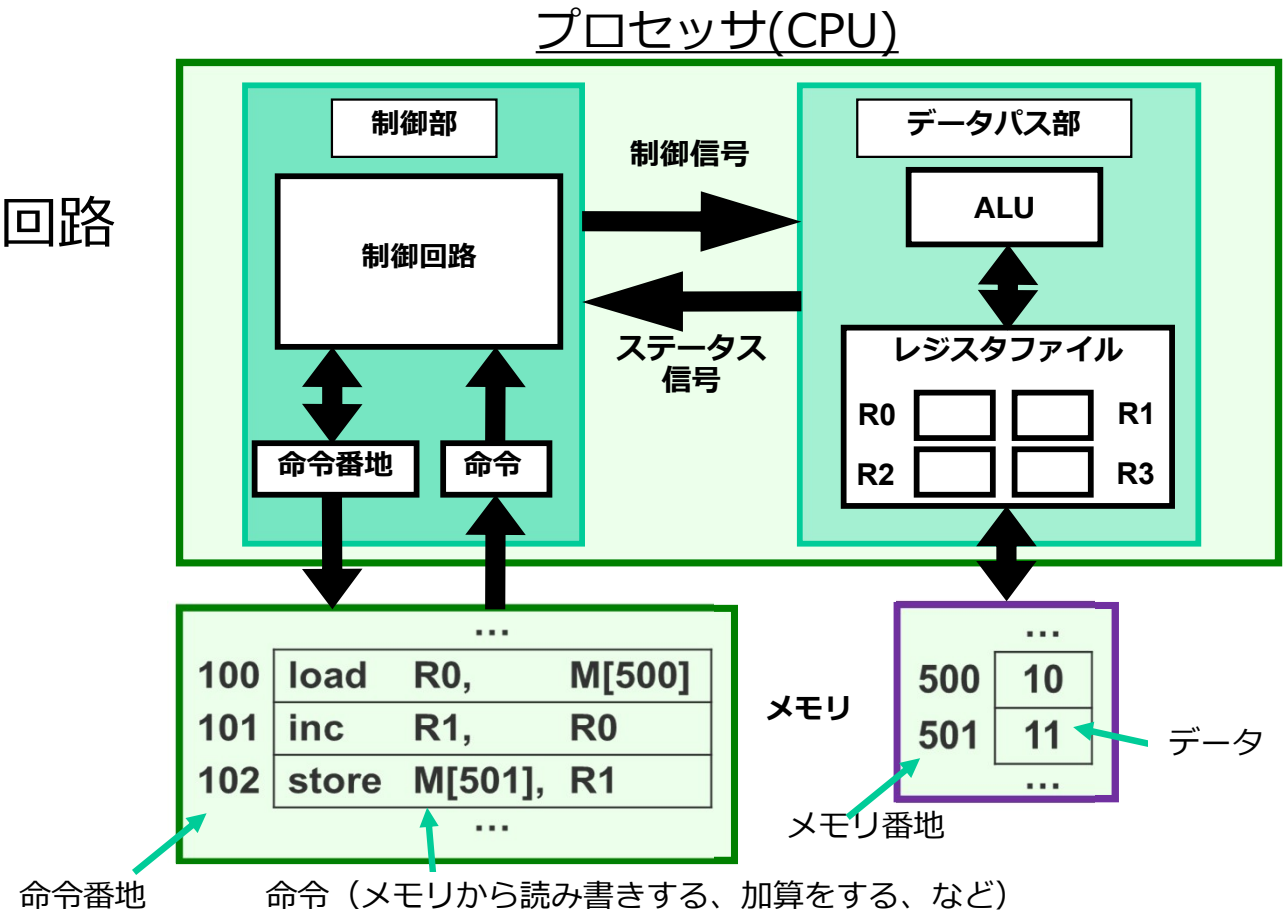
- プログラムにより動作する



- プログラム
  - CPUの命令
  - 演算（加算，減算等），メモリへの書き込み，メモリの読み込み
- フォンノイマン型コンピュータ（プログラム内蔵方式）
  - 命令を読み込み（fetch），解釈（decode），実行（execute）
  - プログラムを変更することで様々な処理が可能

プロセッサ(CPU)：詳細

- 制御回路
  - プログラムに従ってプロセッサを制御する回路
- 算術論理演算器 （ALU）
  - 様々な計算を行う回路
- レジスタファイル
  - データを一時的に格納する回路
  - 複数存在

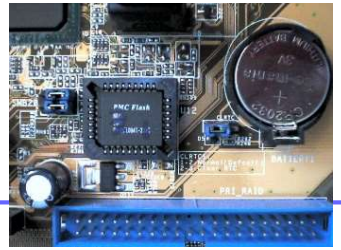


メモリ：RAMとROM

- プログラムやデータを格納する場所
  - 2進数(0か1)のデータ（デジタルデータ）を保持することが可能
- メモリの種類は大きく2種類に分けられる
  - RAM(Random access memory)
    - 元々は格納されたデータに任意の順序でアクセスできる（ランダムアクセス）メモリの意味
    - 現在は，高速に読み書き可能だが，電源が切れるとデータが消えるメモリの意味で使われることが多い
    - DRAMやSRAMが主流

https://commons.wikimedia.org/wiki/File:DRAM_DDR2_512.jpg 2019年3月18日

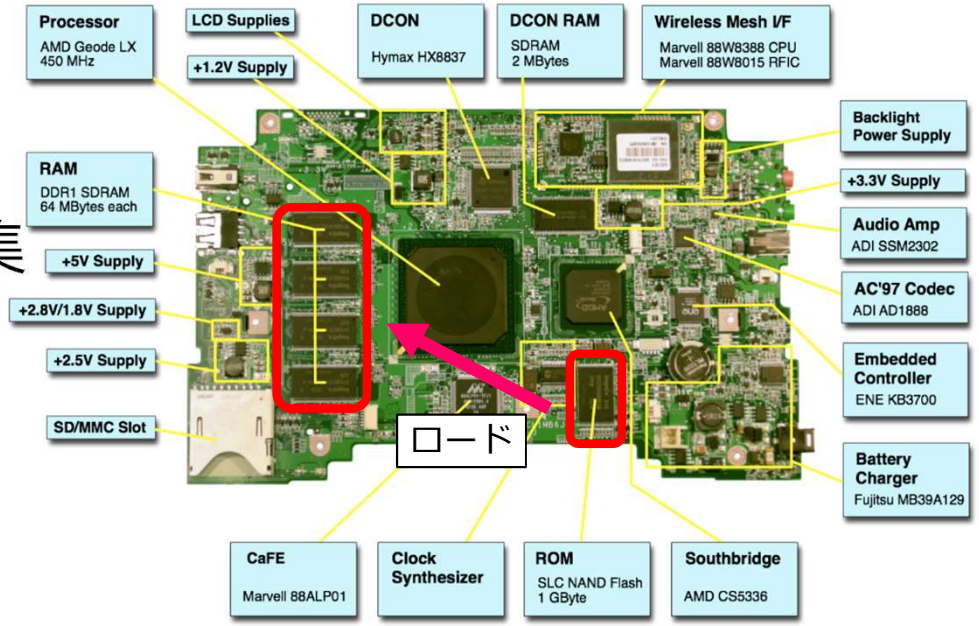
  - ROM(Read only memory)
    - 元々は読み込みのみ可能な（プロセッサから書き込みはできないが電源を切っても内容が消えない）メモリの意味
    - 現在は，RAMより書き込みは遅いが，電源が切れてもデータが消えないメモリ
    - フラッシュメモリが主流



https://sr.wikipedia.org/sr-el/%D0%91%D0%98%D0%9E%D0%A1 2019年3月18日

メモリ：RAMとROMの使い方

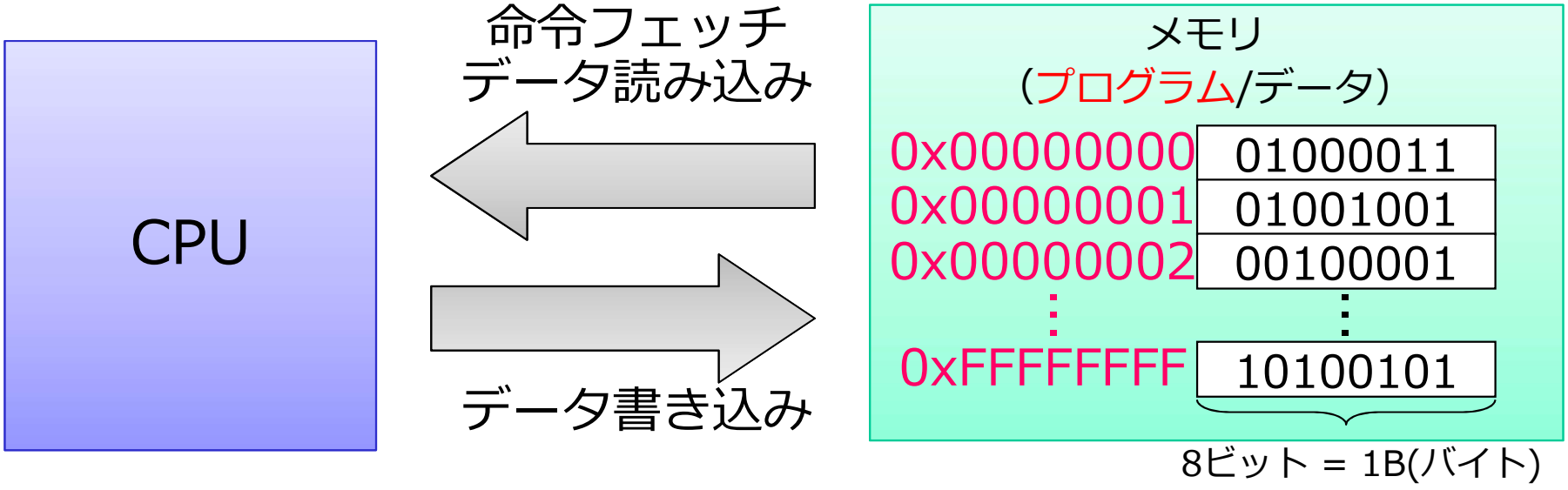
- ROM
  - –プログラム（アプリ）を保存するために使用する
  - –プログラムが使用するデータを保存するために使用する
    - 写真，文章，ゲームのセーブデータ
- RAM
  - –プログラム（アプリ）をROMから読み込んでプロセッサで実行する（プロセッサから読み込む）
  - –データをROMから読み込んでプロセッサから使用する
    - 例)カメラで撮影した画像を編集するために読み込む
  - –プログラム（アプリ）が一時的なデータを書き込む



メモリ：アドレス

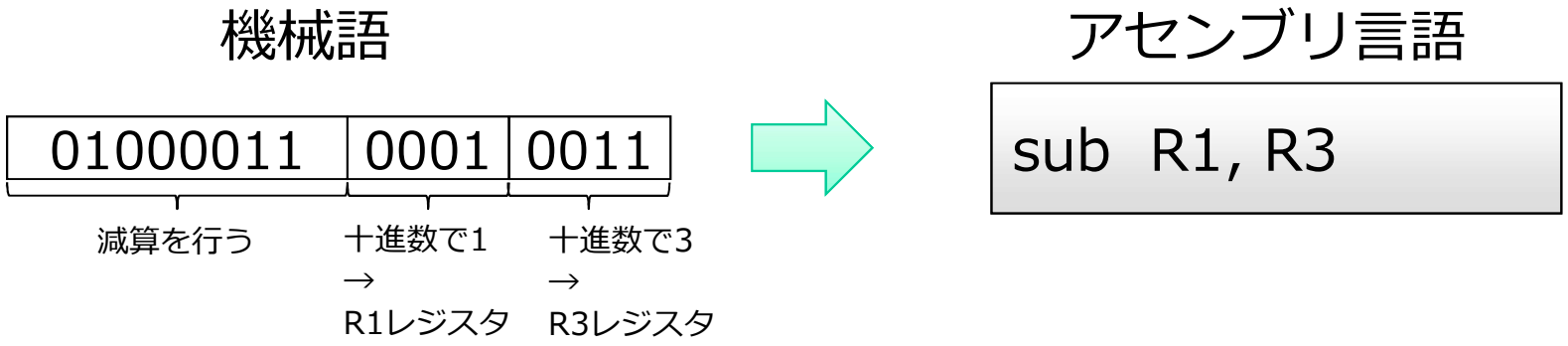
メモリにはアドレス（番地）が割付けられている

- プロセッサはアドレスを指定してメモリを読み書きする
- 1バイト(8ビット)単位でアドレスが割り当てられている
- アドレスの幅により扱えるメモリの最大値が決まる
  - 16bit幅の場合：約64KB(バイト)：約6万4千
  - 32bit幅の場合：約4GB(バイト)　：約40億



プログラム：機械語とアセンブリ言語

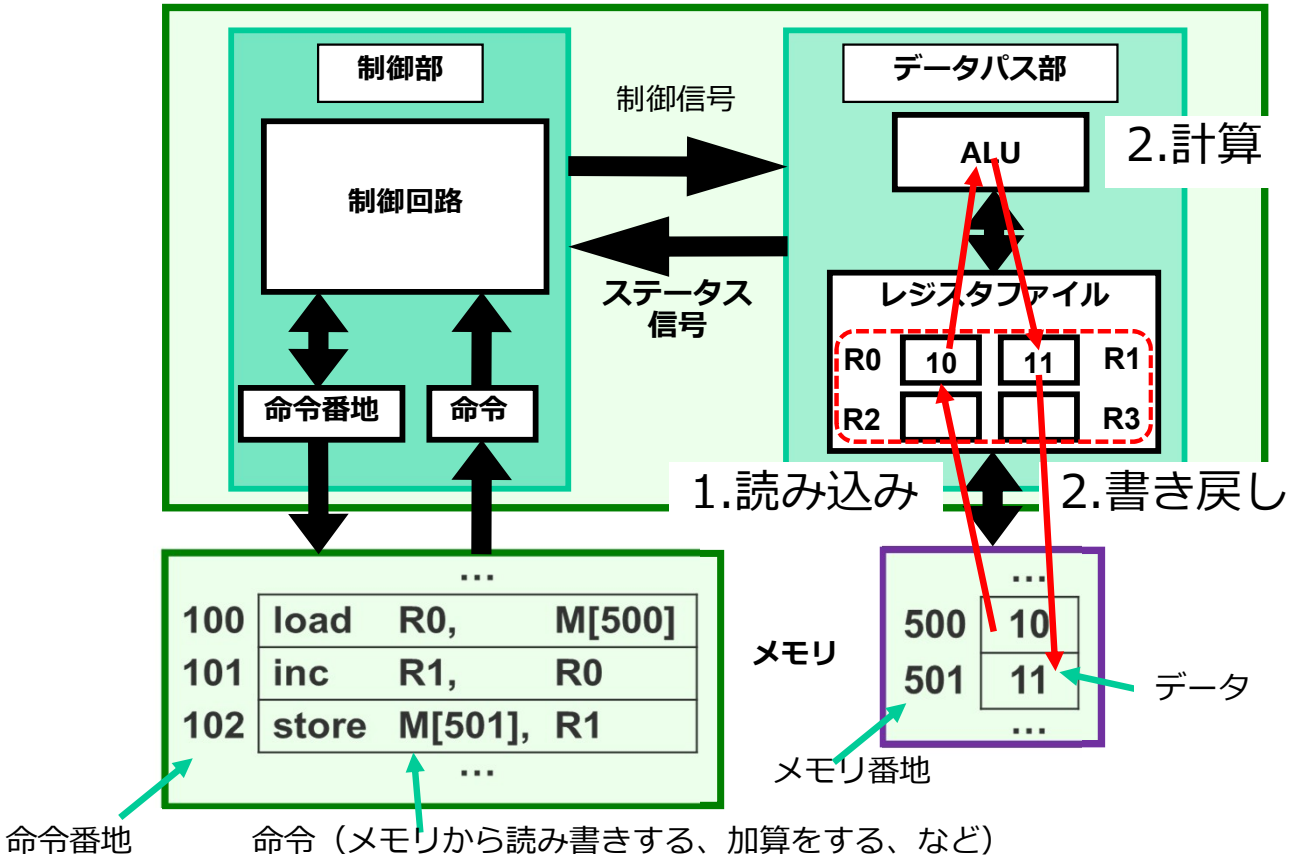
- メモリに置かれているCPUが直接実行可能な命令を機械語と呼ぶ
  - 機械語を組み合わせてプログラムを作成する
- 機械語は人間にとって分かりにくいのでアセンブリ言語と呼ばれる記法でプログラムを記述する
  - 機械語とアセンブリ言語は1対1に対応している
  - 例)レジスタR1からレジスタR3の内容を引いた結果をレジスタR1に格納するプログラム



プロセッサ：レジスタ

プロセッサ内部でのデータの保持や，プロセッサの
動作状態の保持・変更のための変数

- プログラムのデータ（変数）はメモリの中に置かれる
- プロセッサはメモリの値を直接計算できない
- そのため一度レジスタに値を読み，計算して，結果を書き戻す

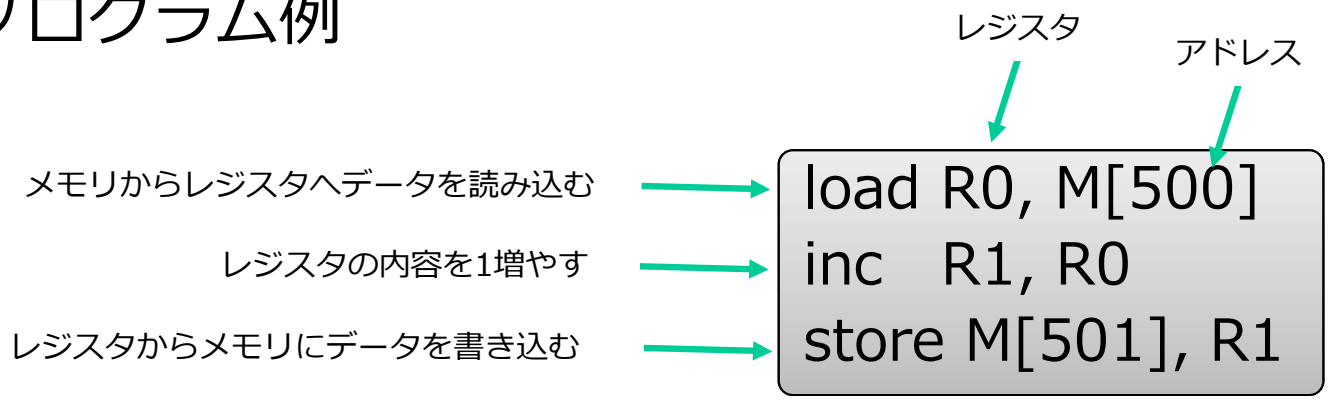


プロセッサ：実行例

機械語（アセンブリ言語）の実行例について説明

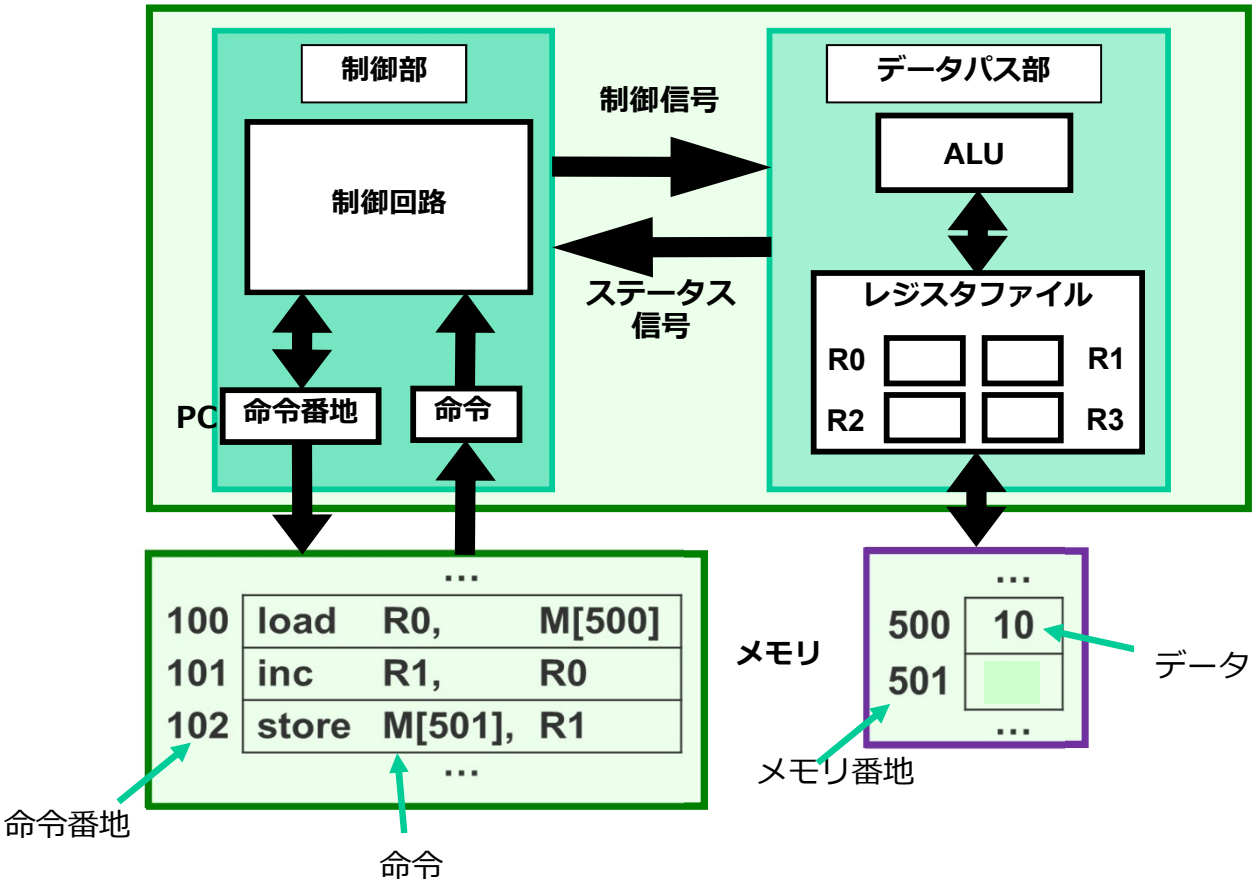
- 実現したい処理
  - メモリから数値を読み込んで1を加算してメモリに書き戻す
    - 読み込むメモリのアドレス：500番地
    - 書き込むメモリのアドレス：501番地

- プログラム例



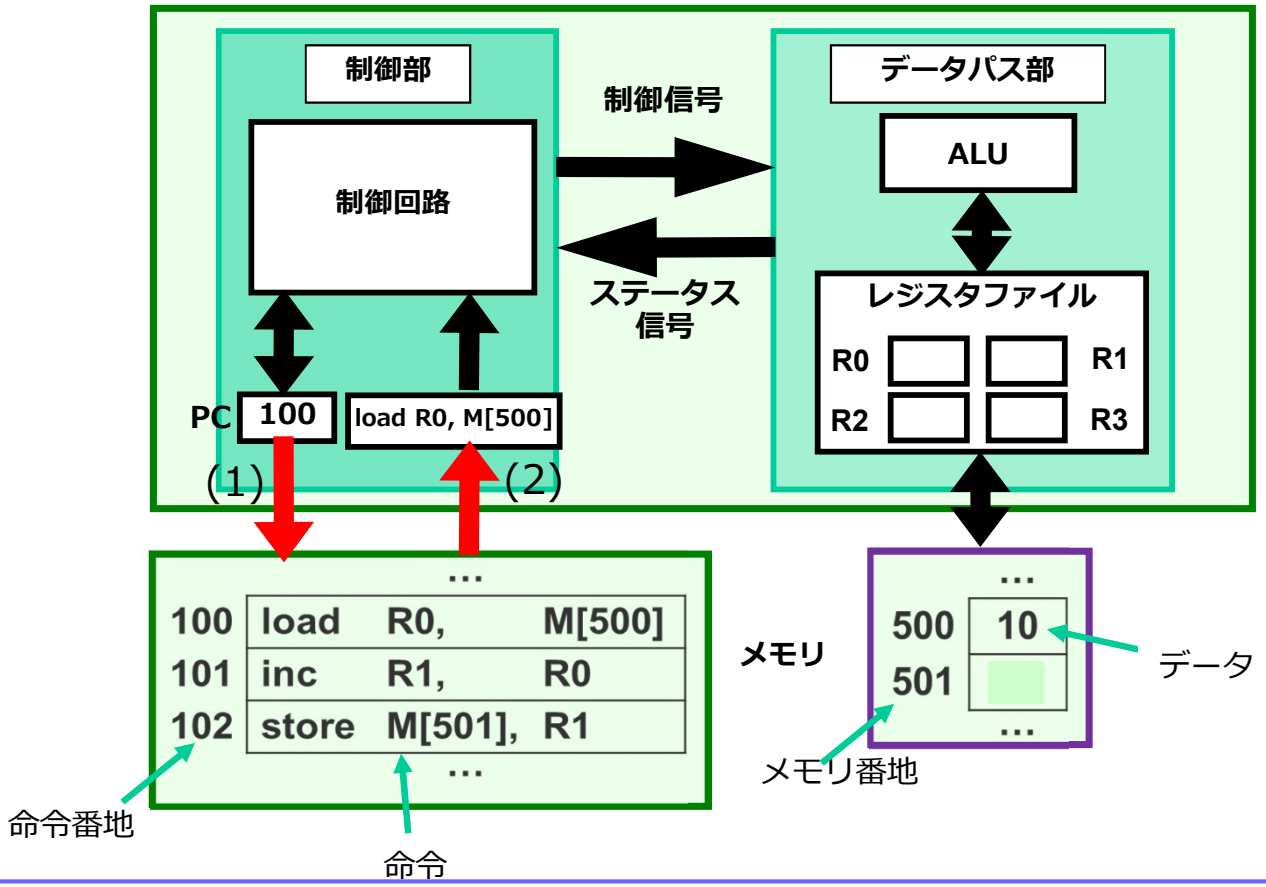
プロセッサ：実行例

- プログラムとデータをメモリに配置する
- プログラムは100番地から配置
- 実行するプログラムの番地はPC(Program Counter)というレジスタが保持



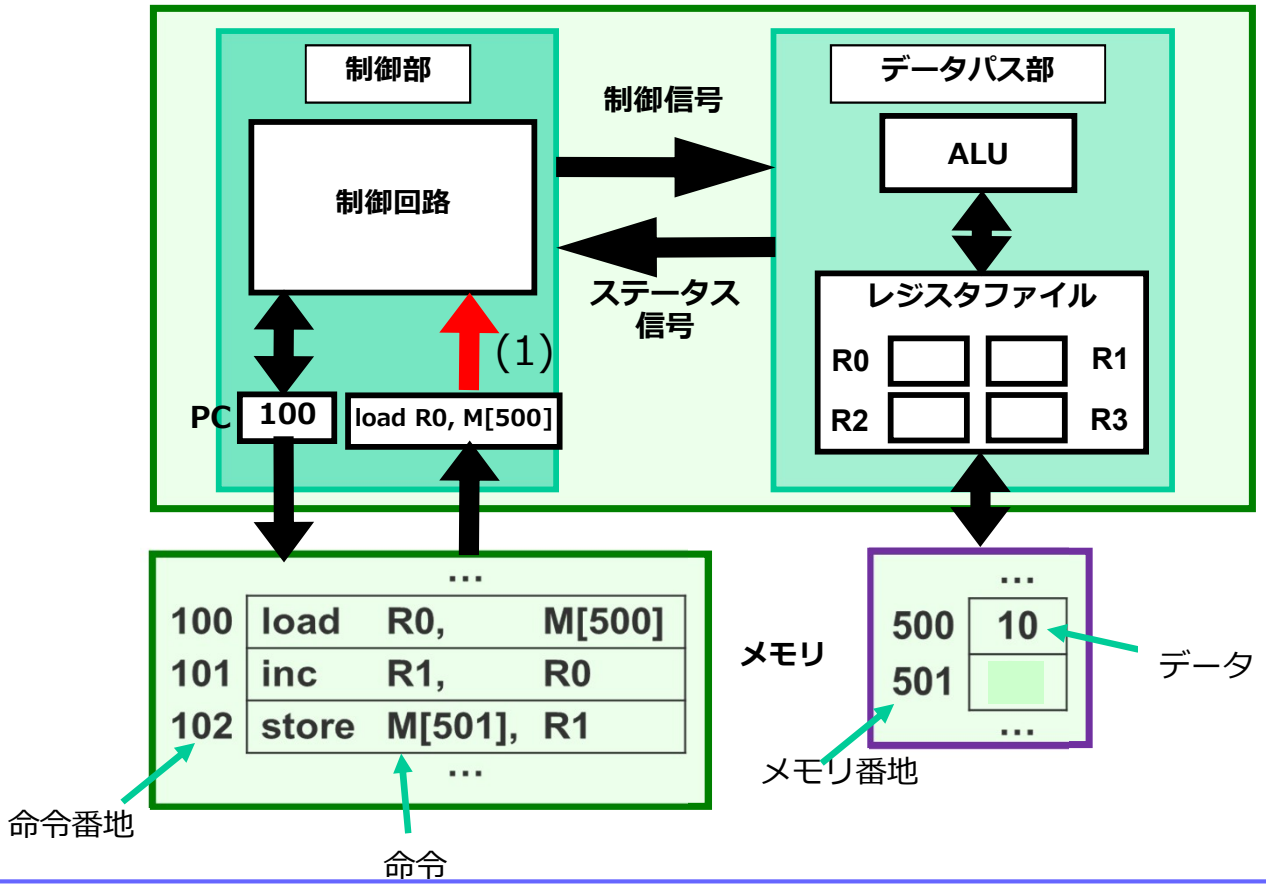
プロセッサ：実行例

- 命令の読み込み
  - –PCに格納されている番地から命令を読み込む
    (1)プロセッサが100番地を読み込むことをメモリに伝える
    (2)メモリは100番地のメモリをプロセッサに渡す



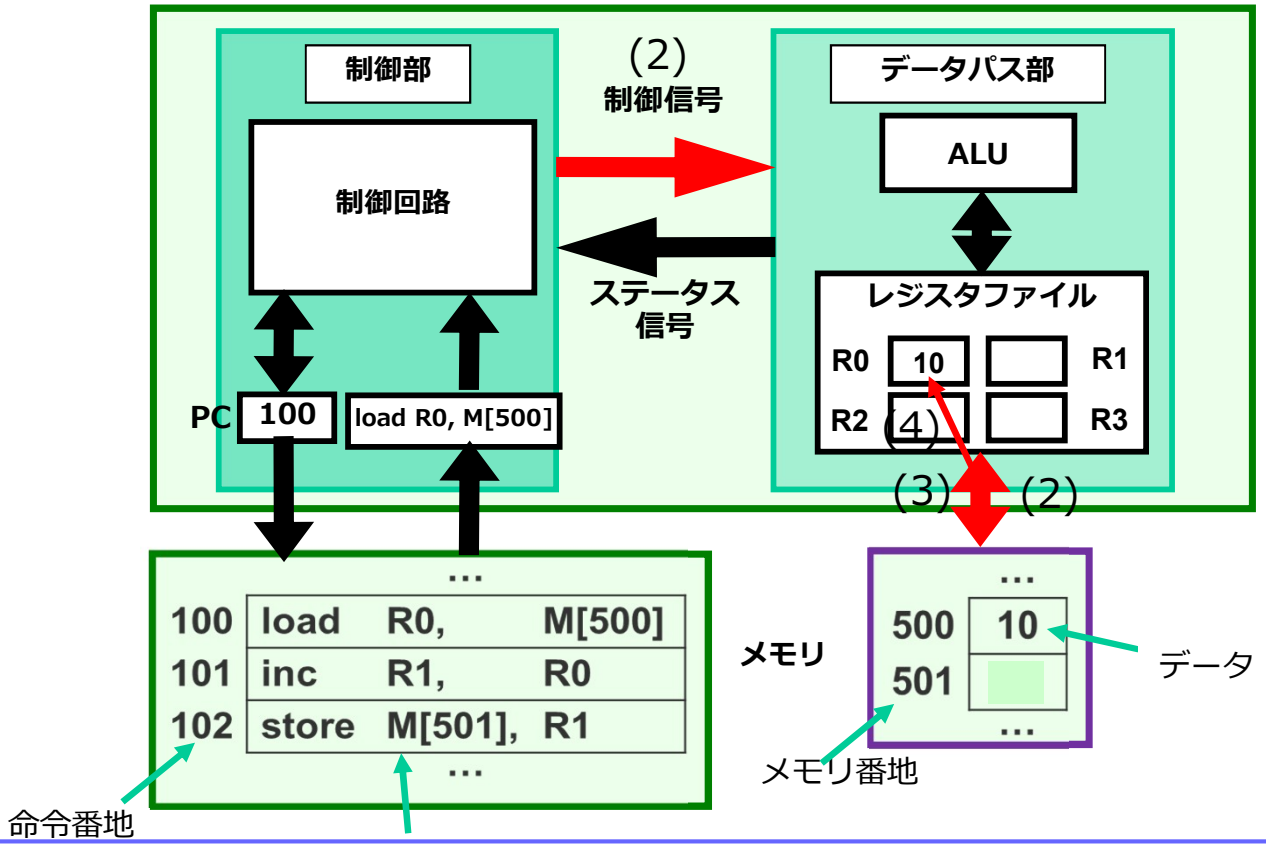
プロセッサ：実行例

- 命令のデコード（解釈）
  - 命令の内容を理解して，行うべき処理を決定する



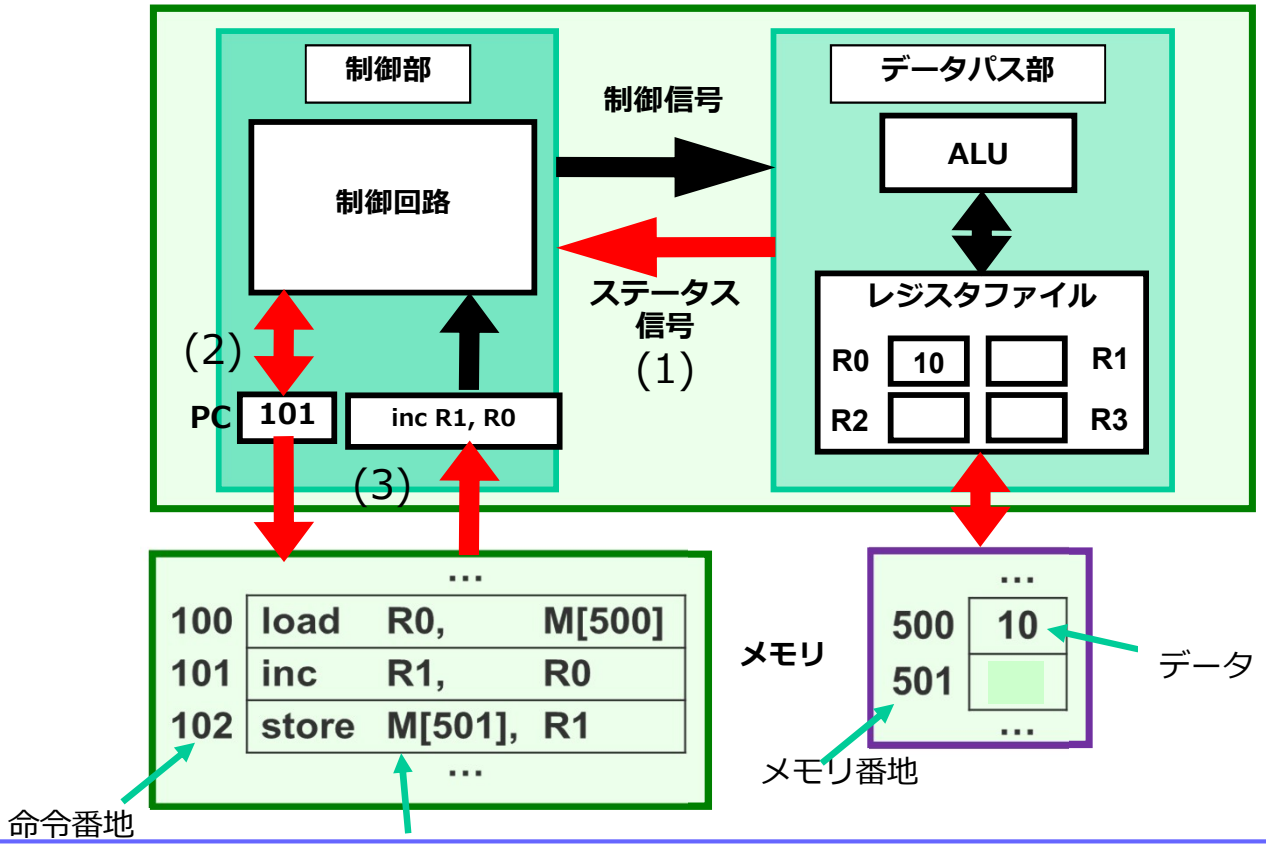
プロセッサ：実行例

- 命令を実行
  - （1）制御回路が制御信号を用いてデータパス部に指令する
  - （2）メモリに500番地を読み込むことを伝える
  - （3）メモリから500番地のデータが渡される
  - （4）渡されたデータをレジスタR0に格納する



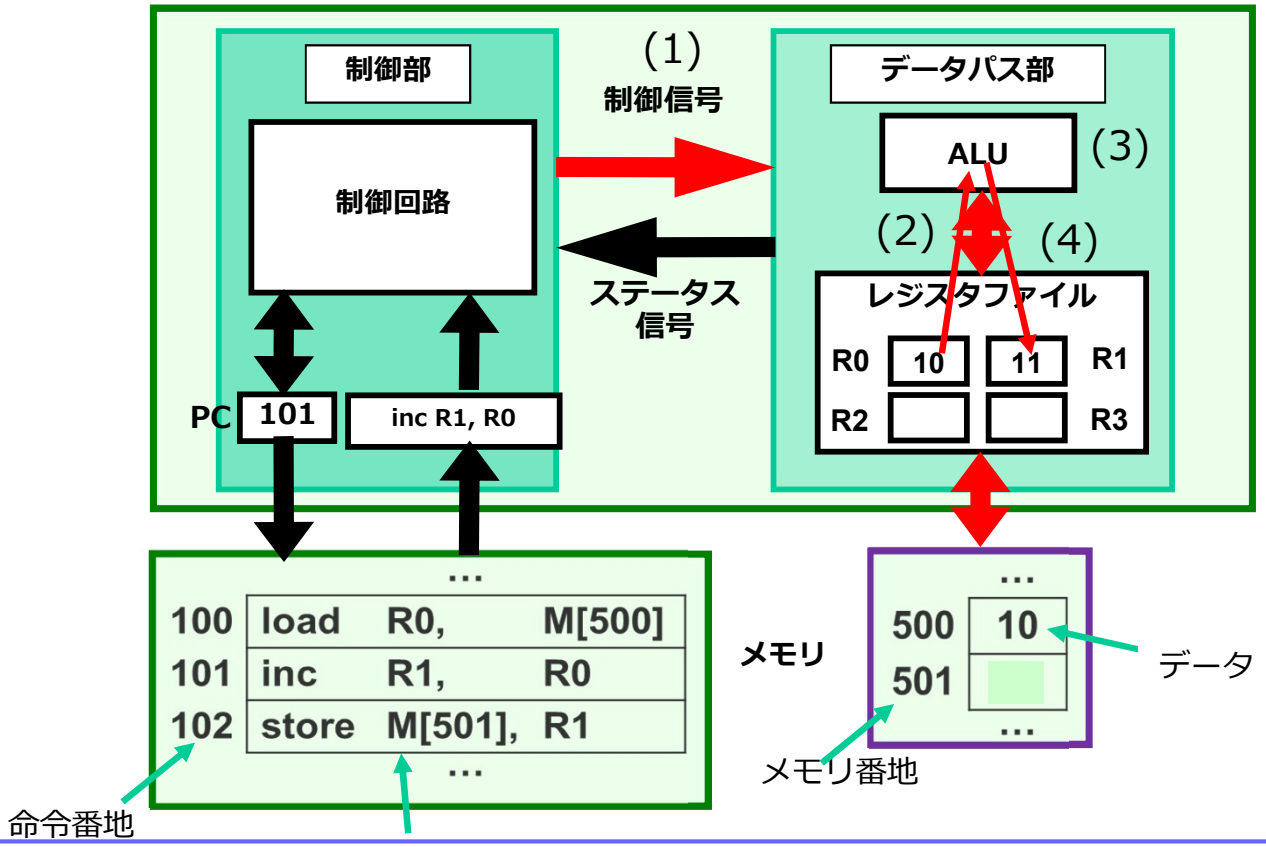
プロセッサ：実行例

- 次の命令の実行に進む
  - （1）データパス部はメモリの読み込みが終了したことをステータス信号で伝える
  - （2）制御回路はPCの値を1増やす
- 次の命令の読み込み
  - （3）101番地の命令を読み込む



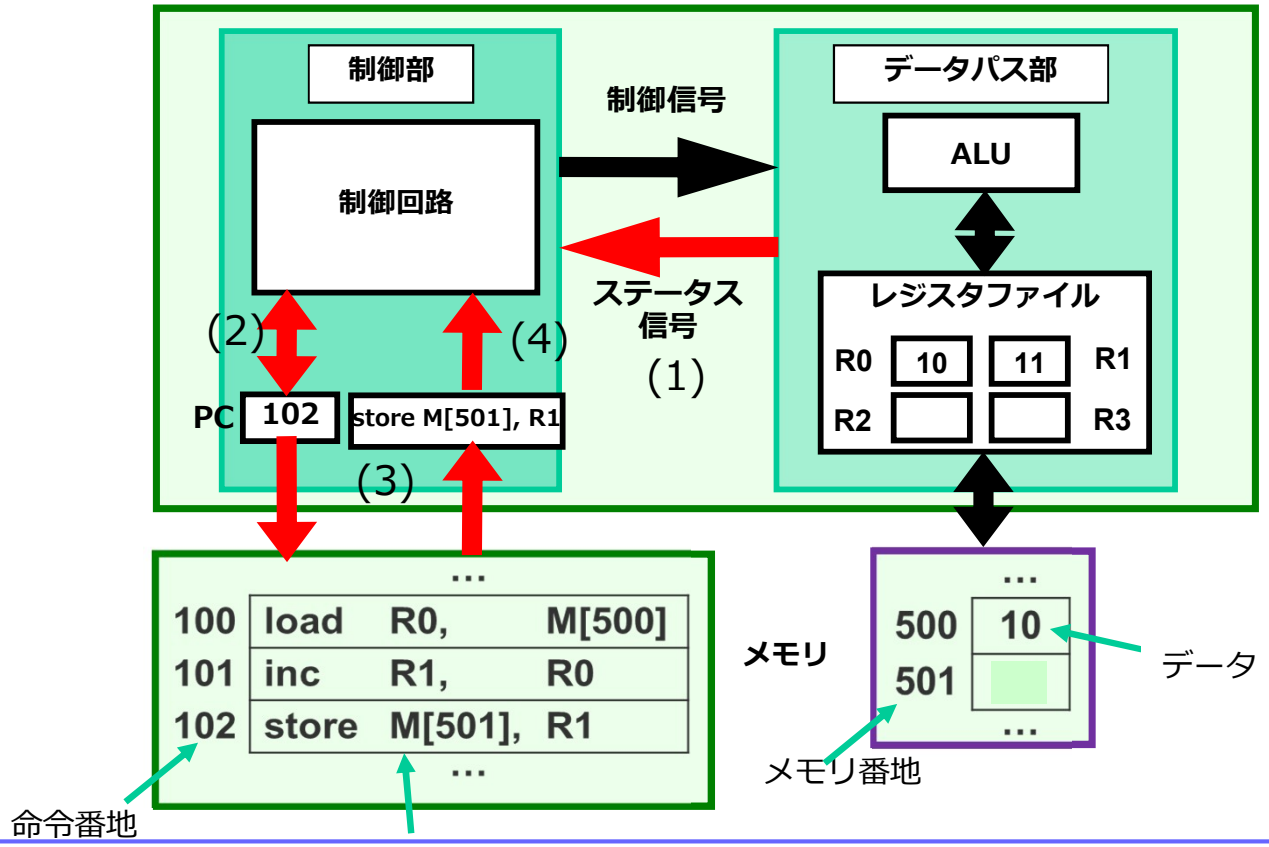
プロセッサ：実行例

- 命令の解釈と実行
  - （1）命令を解釈して制御信号でデータバス部を制御する
  - （2）レジスタR0の内容(10)をALUに読み込む
  - （3）ALUで読み込んだ値に1を加える
  - （4）結果をレジスタR1に格納する



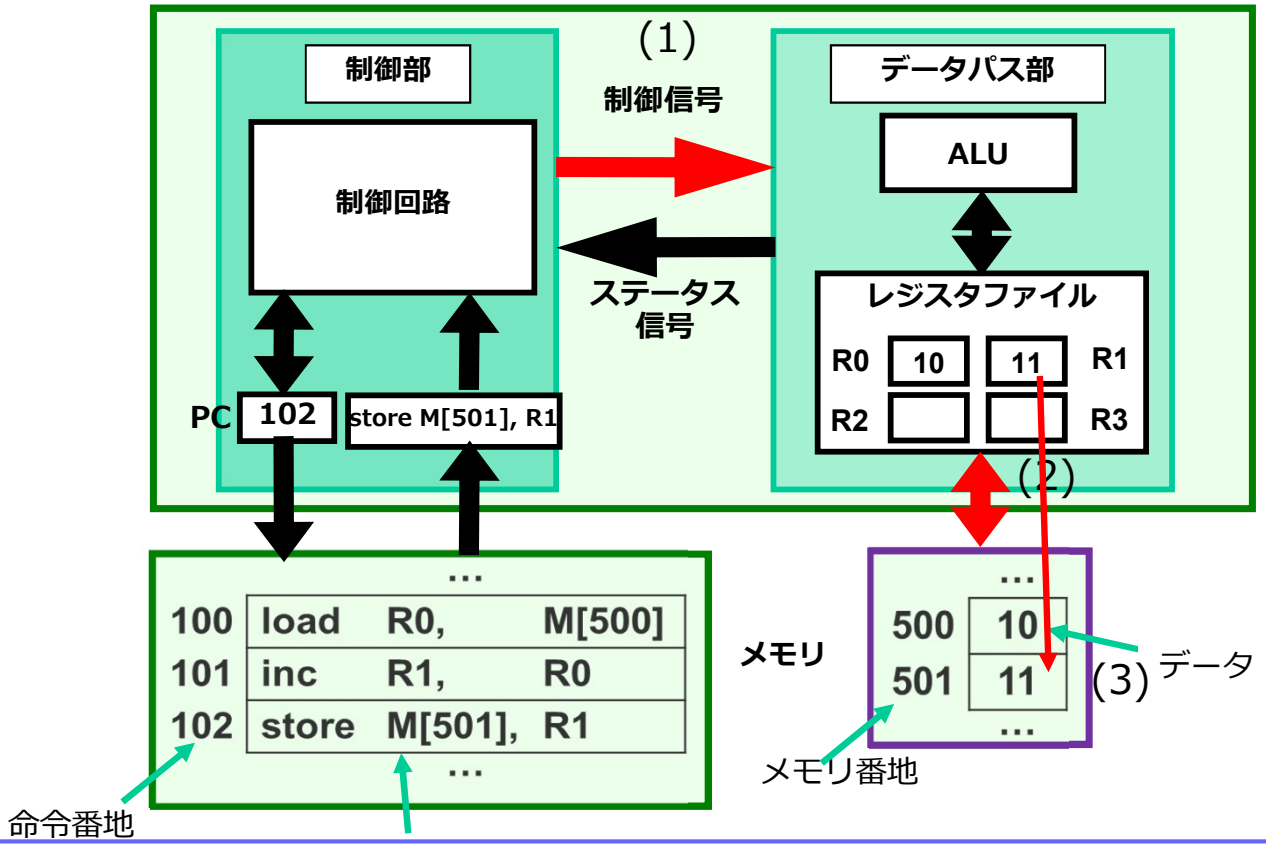
プロセッサ：実行例

（1）制御部に計算の終了を通知
（2）PCを1増やす
（3）102番地の命令を読み込み
（4）命令の解釈



プロセッサ：実行例

- 命令の実行
  - （1）制御回路が制御信号を用いてデータパス部に指令する
  - （2）メモリに501番地に書き込むこと及び書き込むデータを渡す
  - （3）メモリは受け取ったデータを501番地に書き込む



プログラム：命令セット

- あるプロセッサが解釈可能（実行可能）な機械語の命令の集合

- 代表的な命令セット
  - x86/x64　　：PCが実行可能な命令
  - ARM32/ARM64　：スマートフォンが実行可能な命令

```
(a * c) - (b * d)
```

x86命令

```
movl    12(%esp), %eax
movl    16(%esp), %edx
imull   4(%esp), %eax
imull   8(%esp), %edx
subl    %edx, %eax
```

ARM32命令

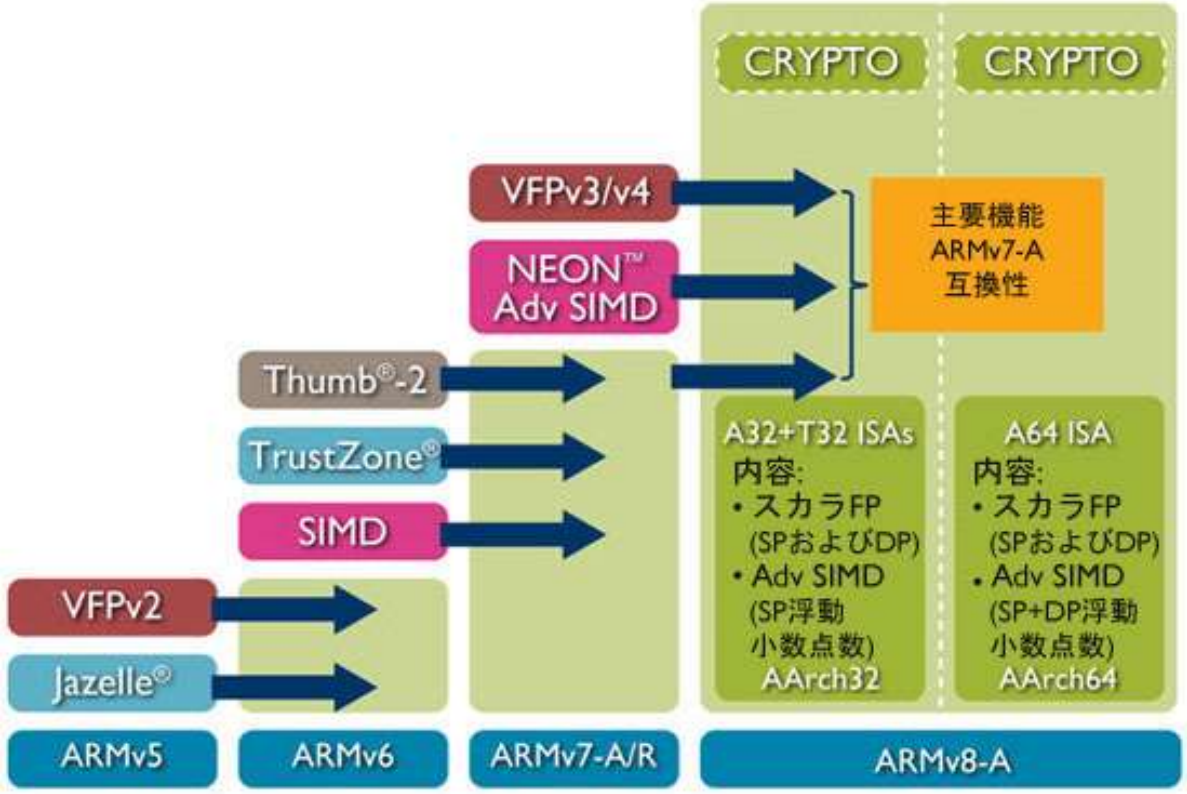
```
mul     r0, r2, r0
mul     r3, r1, r3
rsb     r0, r3, r0
```

ARM64命令

```
mul     w0, w0, w2
msub    w0, w1, w3, w0
```

プロセッサ：命令セットの発展

- 命令セットはアップデートされ命令が追加される（機能が追加される）
  - 逆に命令が減ることは基本的にない（後方互換性）
- 例)ARM命令セットの発展
  - 携帯電話/スマートフォンの機能の増加と共に命令が追加されている
  - Javaの高速実行命令
  - マルチメディア命令
  - セキュリティ命令
  - 64bit命令
  - 仮想化命令



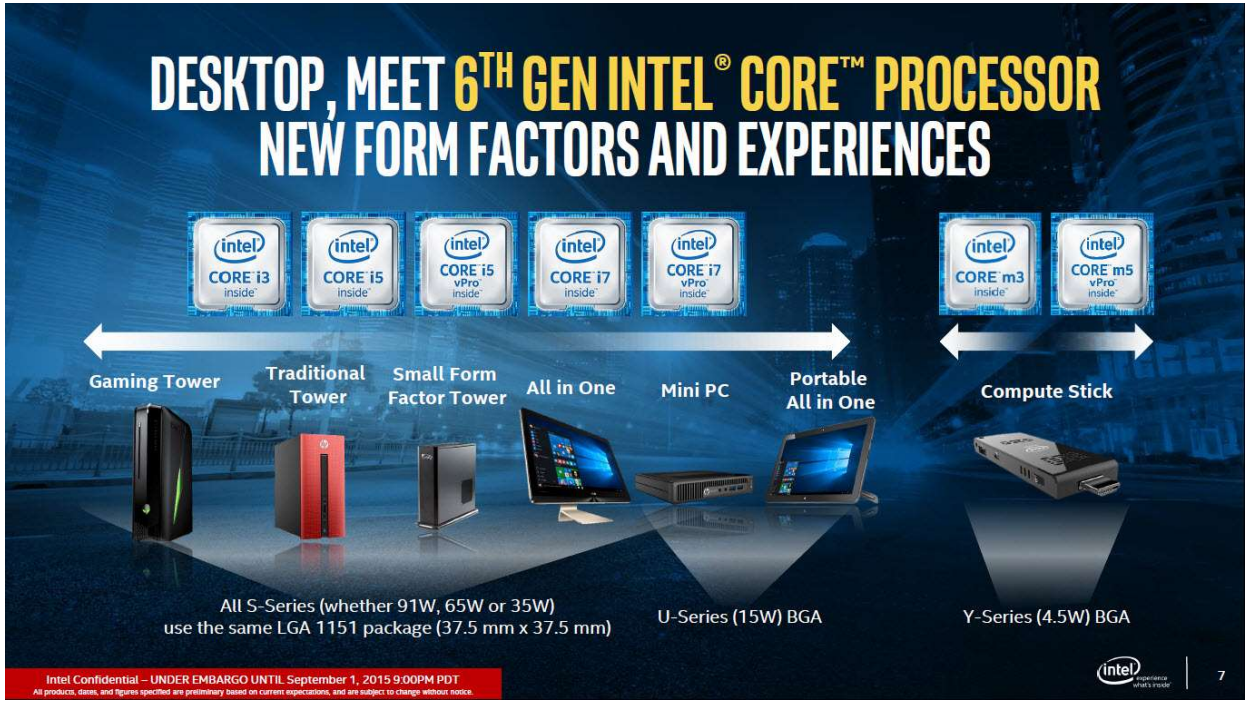
https://pc.watch.impress.co.jp/docs/column/kaigai/689524.html 2019年3月18日

プロセッサ：マイクロアーキテクチャ

- 命令セットを実行するプロセッサ（ハードウエア）の実現方法
- 同一命令セットに対して様々なマイクロアーキテクチャが設計される
  - 特定の命令セットを実行するという機能は同じであるが，他の目的（目標）が異なるため実現方法は様々
- x86/x64の例

デスクトップ・ノート・スティック

http://www.techdata.cz/ftp/MARCOM/Intel.pdf 2019年3月18日



サーバー

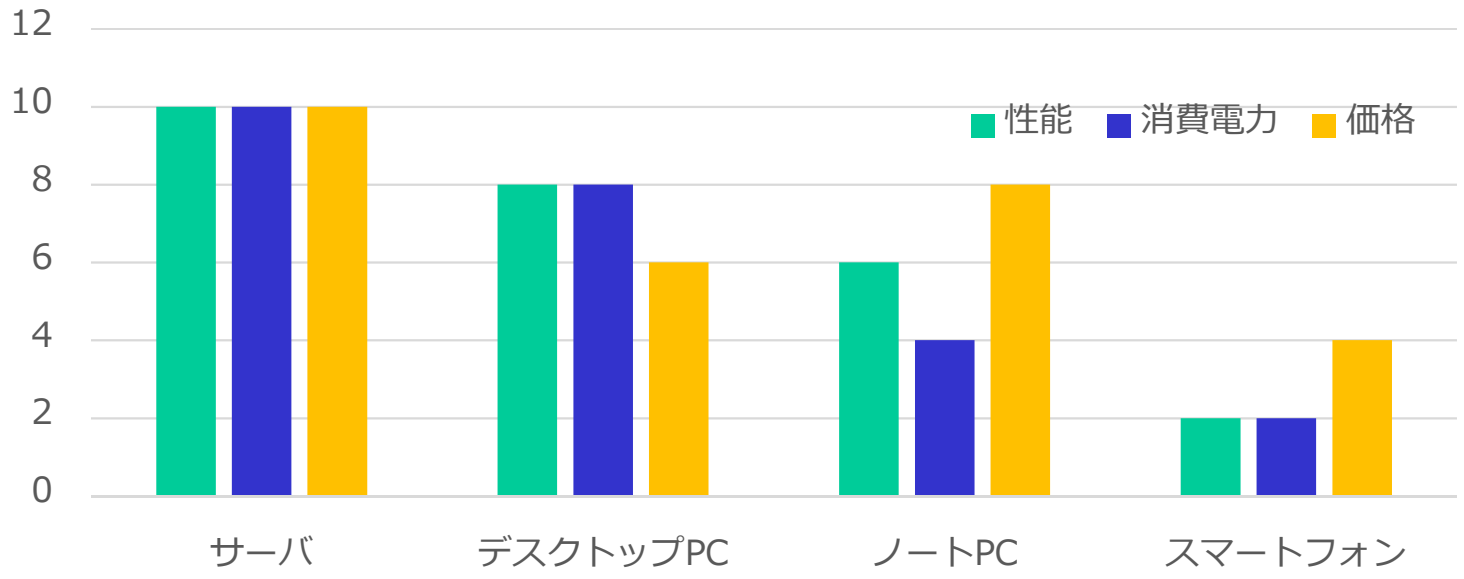
https://wdplu.com/xserverphp7 2019年4月24日

https://www.amazon.co.jp/dp/B00116XB6W
2019年4月24日

https://www.amazon.co.jp/dp/B00FKG9R2Q 2019年4月24日

プロセッサ：マイクロアーキテクチャ

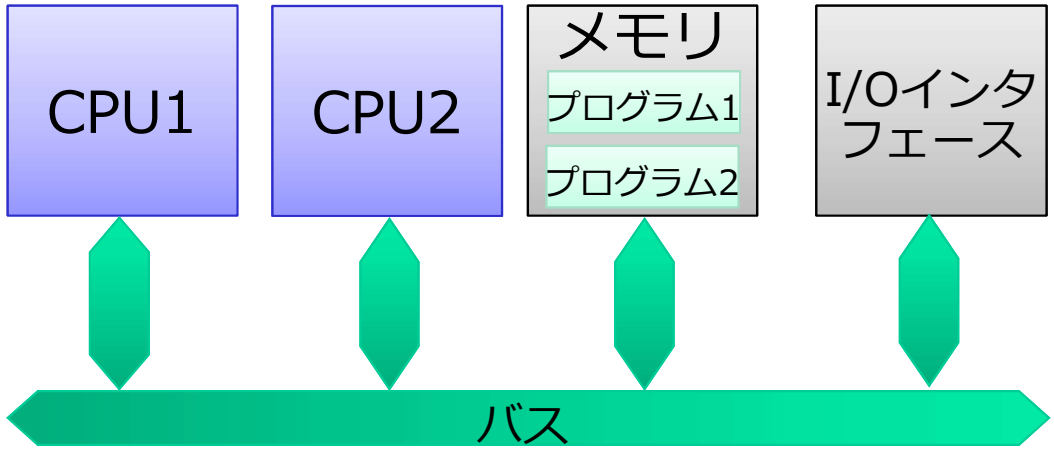
- プロセッサ設計時の目標
  - 理想的なプロセッサ
    - 高速で消費電力が少なく安価
  - 全ての目標を高く実現することは困難
    - 高速化するためには多くのトランジスタが必要
      - ✓消費電力が多くなり値段も上がる
  - 応用毎にそれぞれのトレードオフを決めてアーキテクチャを決定



マルチプロセッサ

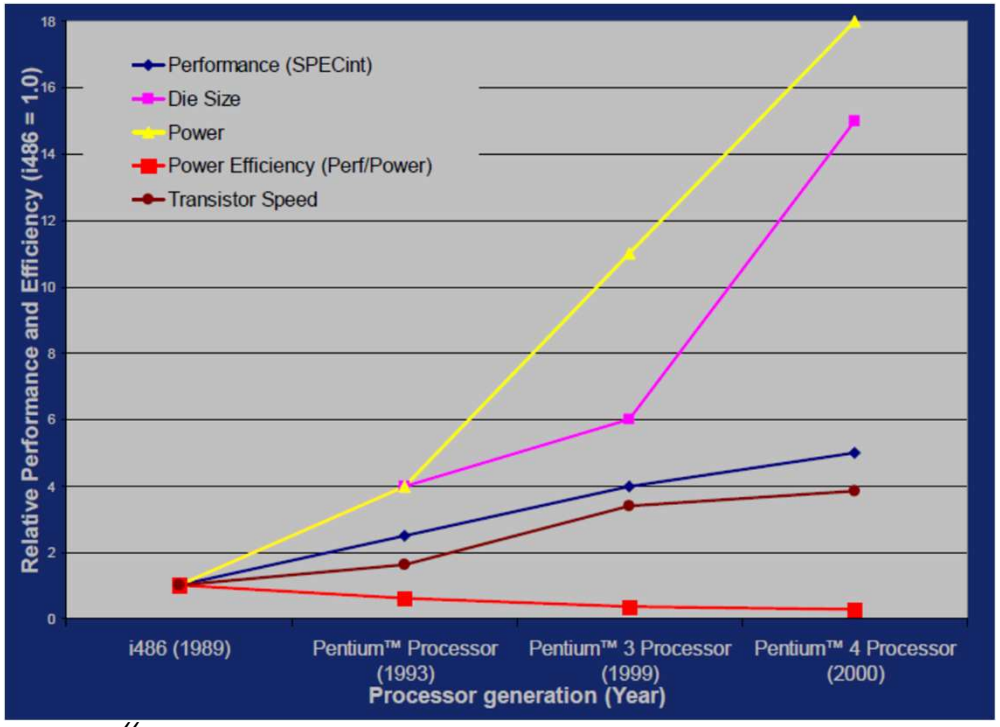
- 複数のプロセッサを使用するコンピュータ
  - ノートPC　　：2～4コア
  - デスクトップPC　：4～8コア
  - サーバー　　：4～20コア
  - スマートフォン　：2～10コア

- プログラム
  - それぞれのプロセッサで独立してプログラムを実行



マルチプロセッサ

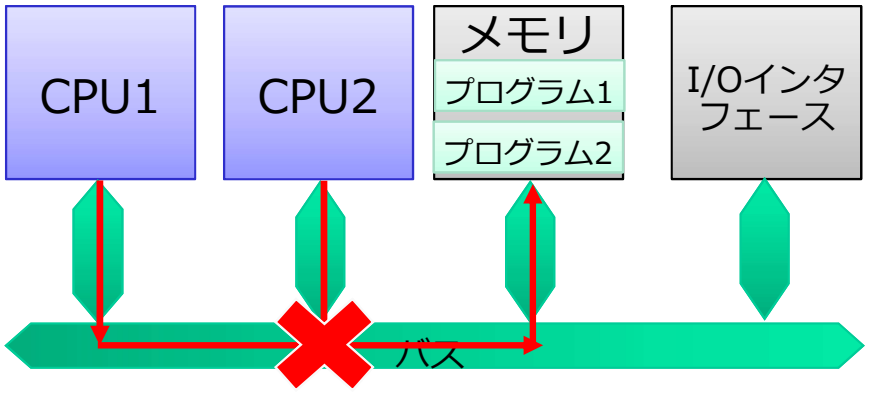
- なぜマルチプロセッサになっているのか？
  - PCは2005年頃まではシングルコアが主流
  - その後，動作周波数（ドランジスタのスイッチ速度）が上げられなくなった（発熱が大きく冷却できない）
  - 単体のプロセッサの性能を上げられなくなった
  - ⇒ 複数のプロセッサを用いることで性能を上げる方向に変更



Source: Chris Rowen, “The Reinvention of the Microprocessor for MPSOC,” MPSoC 2006.

マルチプロセッサ：スケーラビリティ

- プロセッサがn個になれば性能はn倍になるか？
  - 様々な要因によりコア数に比例して性能は上がらない
- ハードウェア
  - 各プロセッサが共通に使用する部分により性能が上がらない
  - 例)メモリへの読み込み・書き込みが衝突して時間がかかる



- ソフトウェア
  - 並列に実行するのが困難な計算がある
    - 並列化容易な計算　：数列の和
    - 並列化が難しい計算：フィボナッチ数列

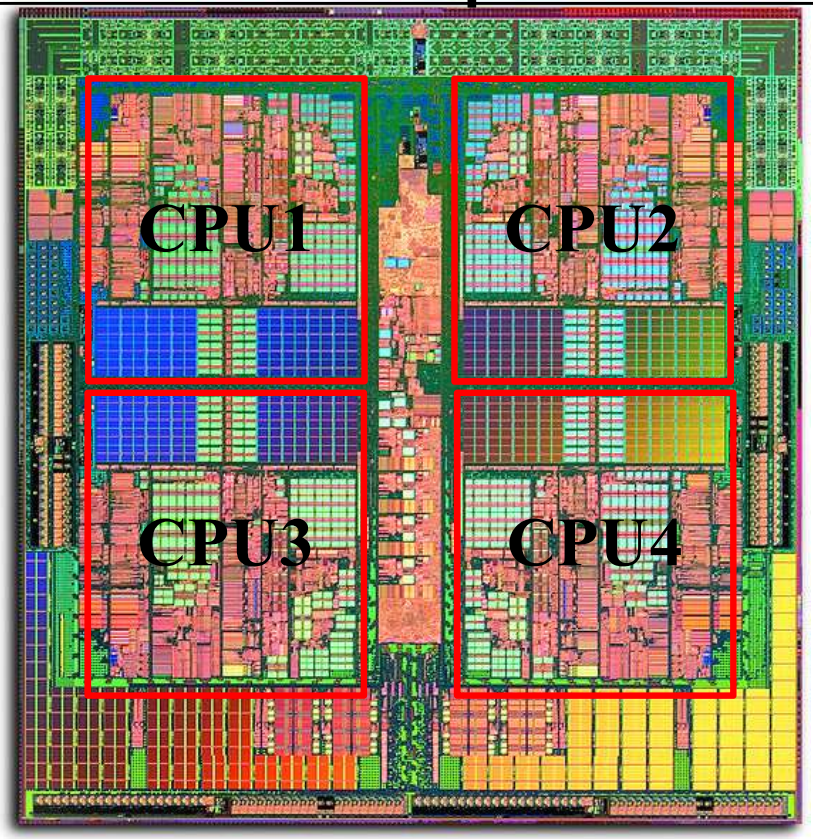
$$\sum_{i=1}^{N} i = \sum_{i=1}^{N/2} i + \sum_{i=N/2}^{N} i$$

$$a_{n+2} = a_{n+1} + a_n$$

マルチプロセッサ：構成

- PCのマルチプロセッサ
  - –同じプロセッサを複数搭載
  - –どのプロセッサでプログラムを実行しても同じ性能

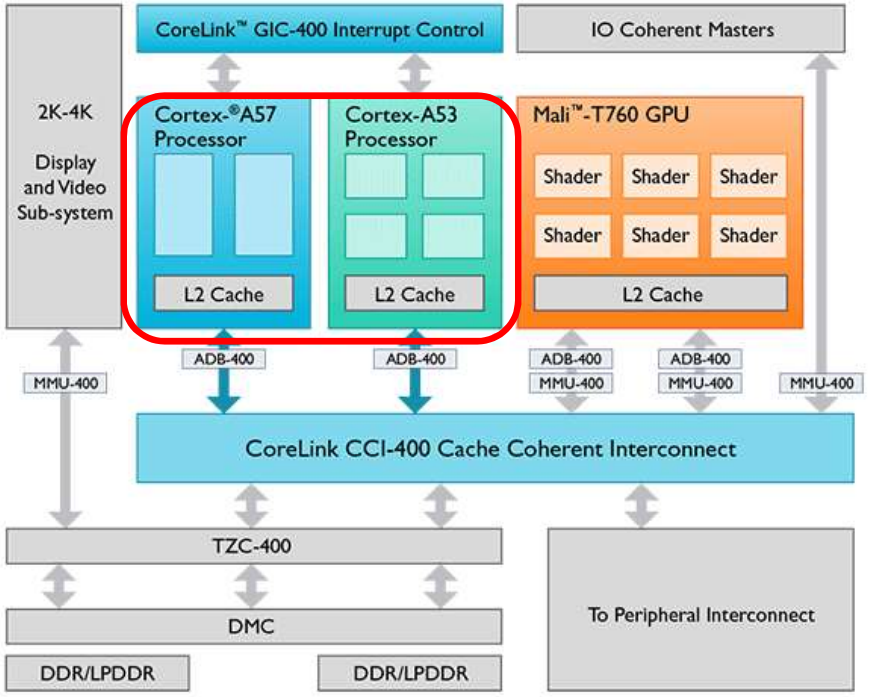
Quad-Core AMD Opteron のダイ



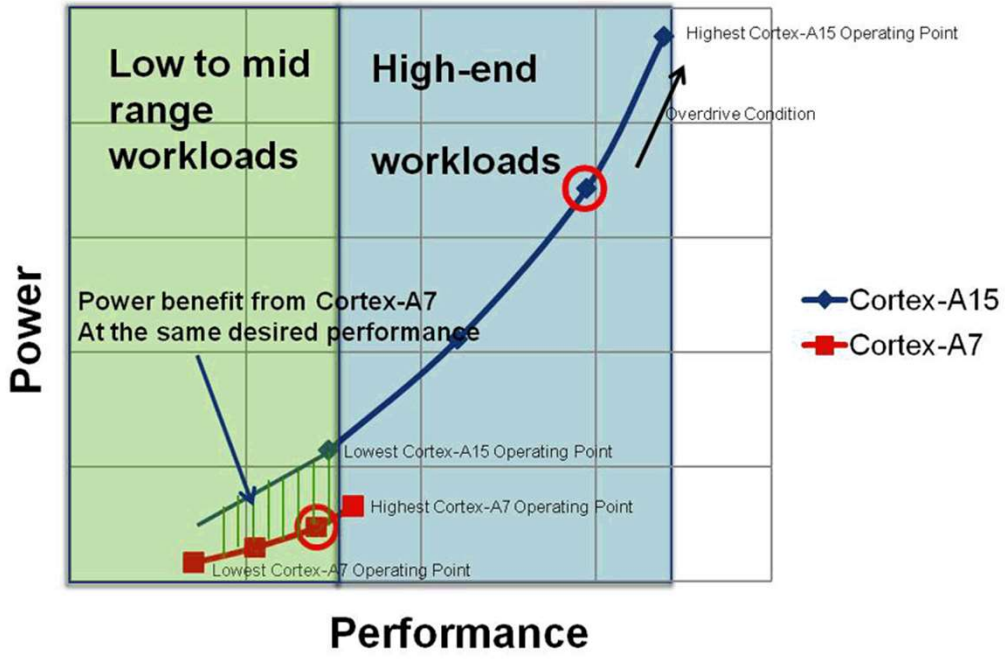
出典：Wikipedia

マルチプロセッサ：構成

- スマートフォンのマルチプロセッサ
  - –性能と消費電力が異なるプロセッサを複数搭載
  - –消費電力の最適化のため，利用状況で使用するプロセッサを変更
    - ゲーム　　：高性能プロセッサで実行
    - Webブラウザ：低性能プロセッサで実行



Pantelis Antoniou (2013)"Adventures in (simulated) Asymmetric Processing"
https://elinux.org/images/4/40/Elc-pantelis-2013.pdf 2019年3月18日

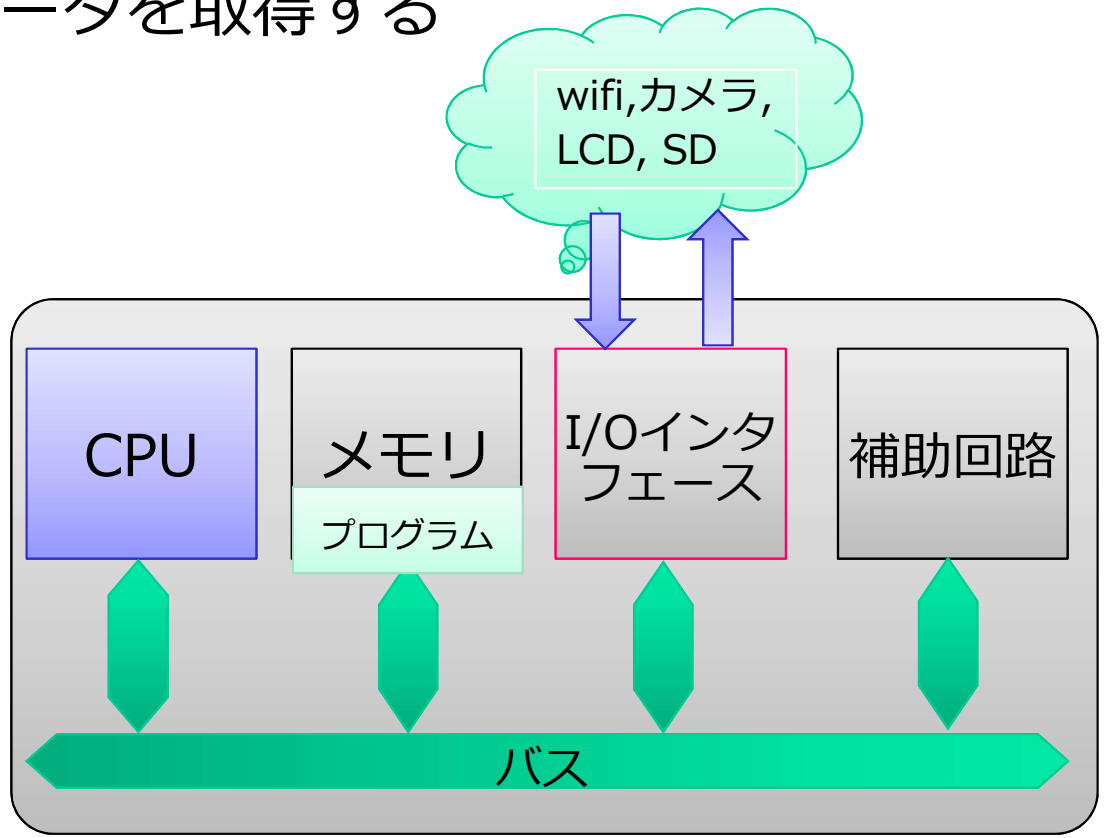


Shaun Purvis, Xilinx (2017) "Heterogeneous Software Architecture with OpenAMP" ESCconf
http://my.presentations.techweb.com/events/esc/boston/2017/conference/download/5385
2019年3月28日

周辺回路：I/Oインタフェース

プロセッサが外界とやりとりをするための回路

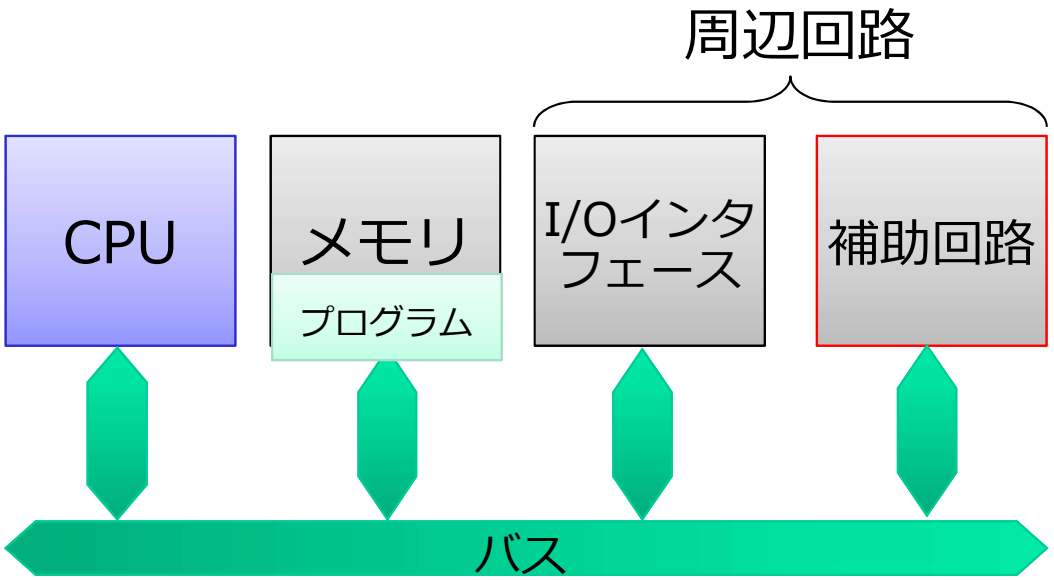
- Wifi経由でインターネットからデータを取得する
  - –Wifiコントローラ
- 画面に写真を表示する
  - –ディスプレイインタフェース
- カメラから画像を取り込む
  - –カメラインタフェース
- SDカードにデータを書き込む
  - –SDカードインタフェース



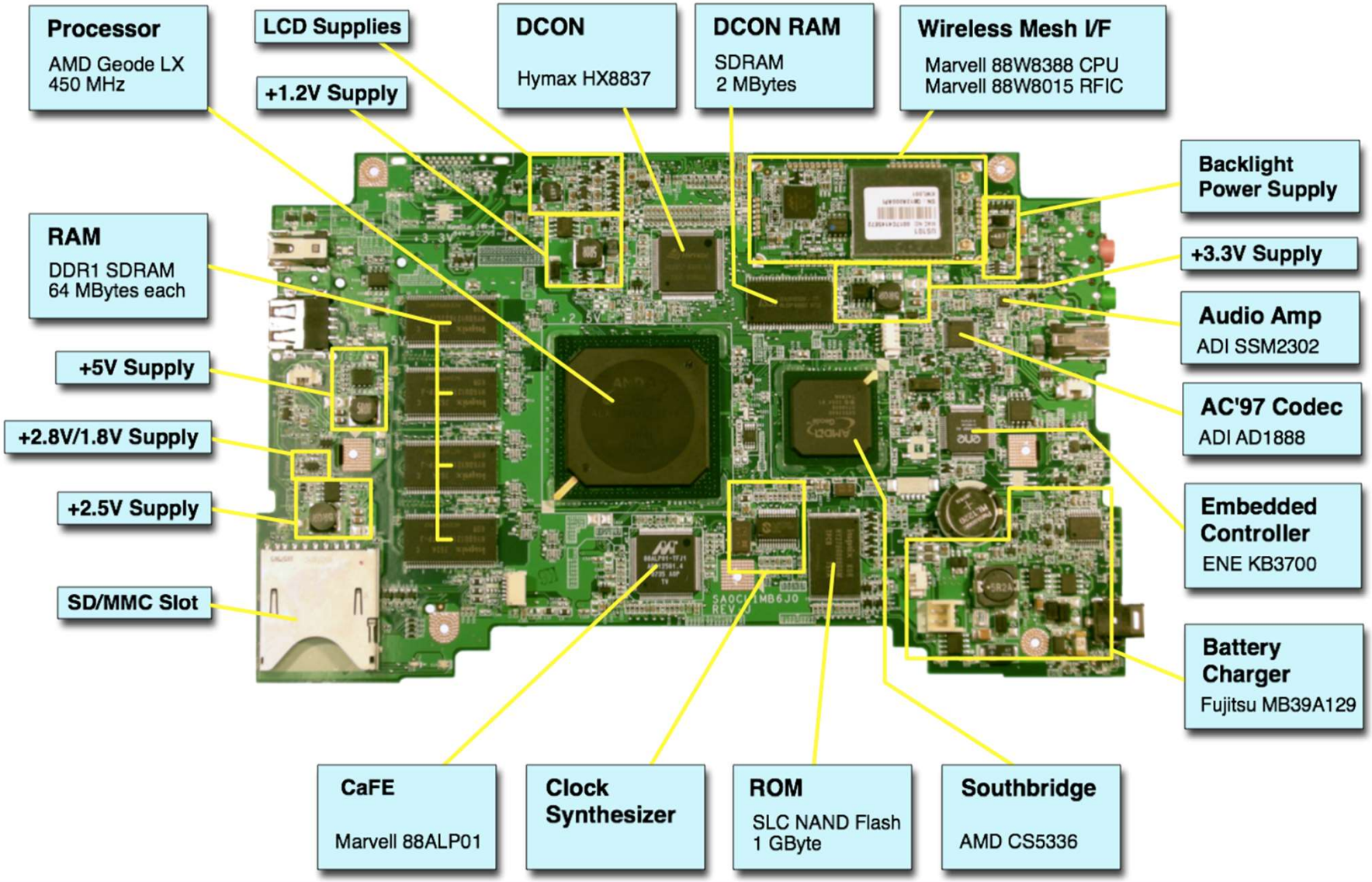
周辺回路：補助回路

プロセッサを機能を補助する回路

- 現在時刻を管理する
  - –リアルタイムクロック
- 時間経過を管理する
  - –タイマ
- データを暗号化する
  - –暗号化モジュール



周辺回路：I/Oインタフェースの例

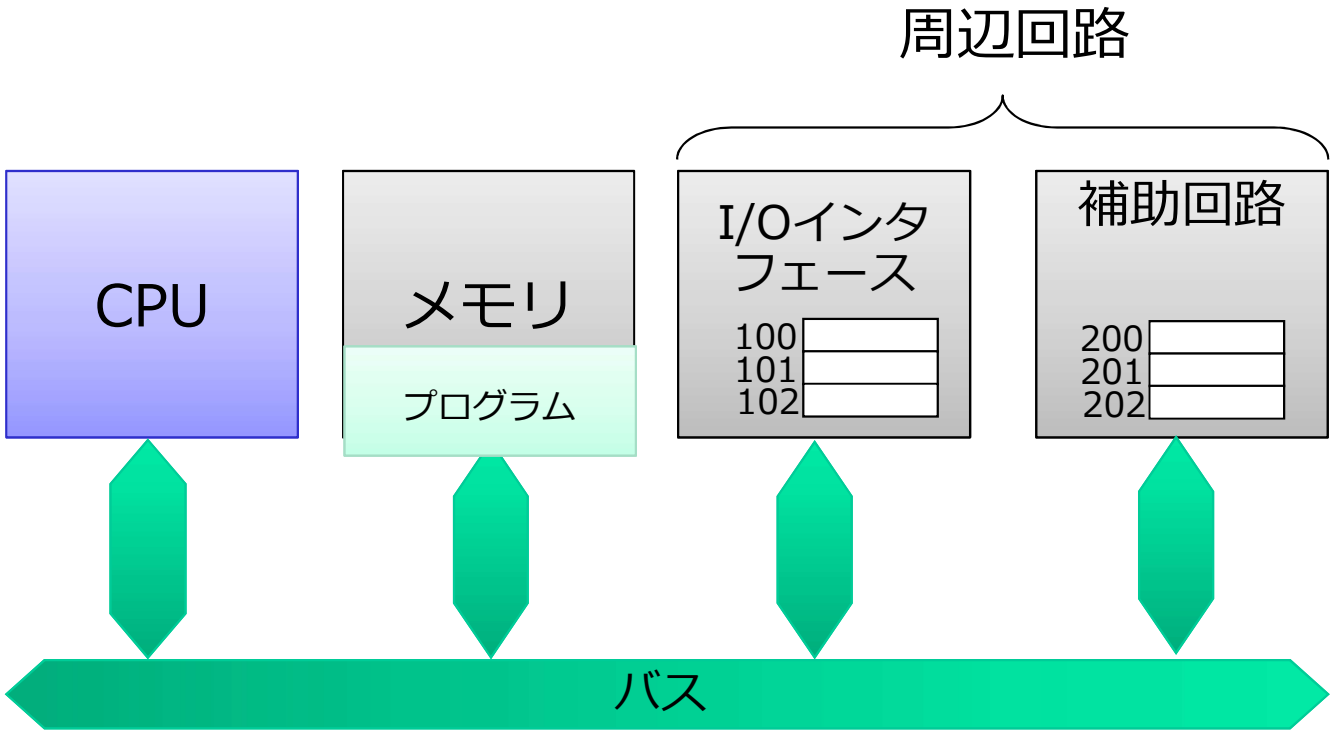


https://ja.wikipedia.org/wiki/OLPC_XO-1 2019年3月18日

周辺回路：デバイスレジスタ

CPUと周辺デバイスとのインタフェース

- 周辺デバイスはそれぞれデバイスレジスタを持つ
- デバイスレジスタにはメモリ同様にアドレスが割り付けられている
- プロセッサからはメモリと同じように読み込みが可能
- デバイスレジスタを使ってプロセッサから周辺回路に指示を出したり，周辺回路からデータを読み込む



プログラム言語

プログラム：高水準言語

- 機械語（アセンブリ言語）のプログラムの問題点
  - ある処理を実現する場合の記述量が大きい（記述の抽象度が低い）
  - 命令セットが異なるプロセッサ間では同じプログラムを実行することができない
    - ⇨ 手間が少なく（記述量を少なく），どのプロセッサでも実行可能なプログラムが書けるようにしたい

- 高水準言語
  - 機械語と比較して，同じ処理を少ない記述量で記述可能かつ，特定の命令セットに依存しないプログラム言語
  - 高水準言語の例
    - C，C++，C#, Objective-C, Java,
    - Python, Ruby, Perl, PHP

プログラム：高水準言語

- C言語
  - 代表的な高水準言語
  - 1972年に最初のリリース

文字列出力するプログラム

```
#include <stdio.h>
int main(void){
    printf("Hello, world!¥n");
    return 0;
}
```

- Python
  - C言語より簡単に記述できる言語
  - 1991年に最初のリリース

```
print "Hello World"
```

プログラム：高水準言語のプログラムの実行

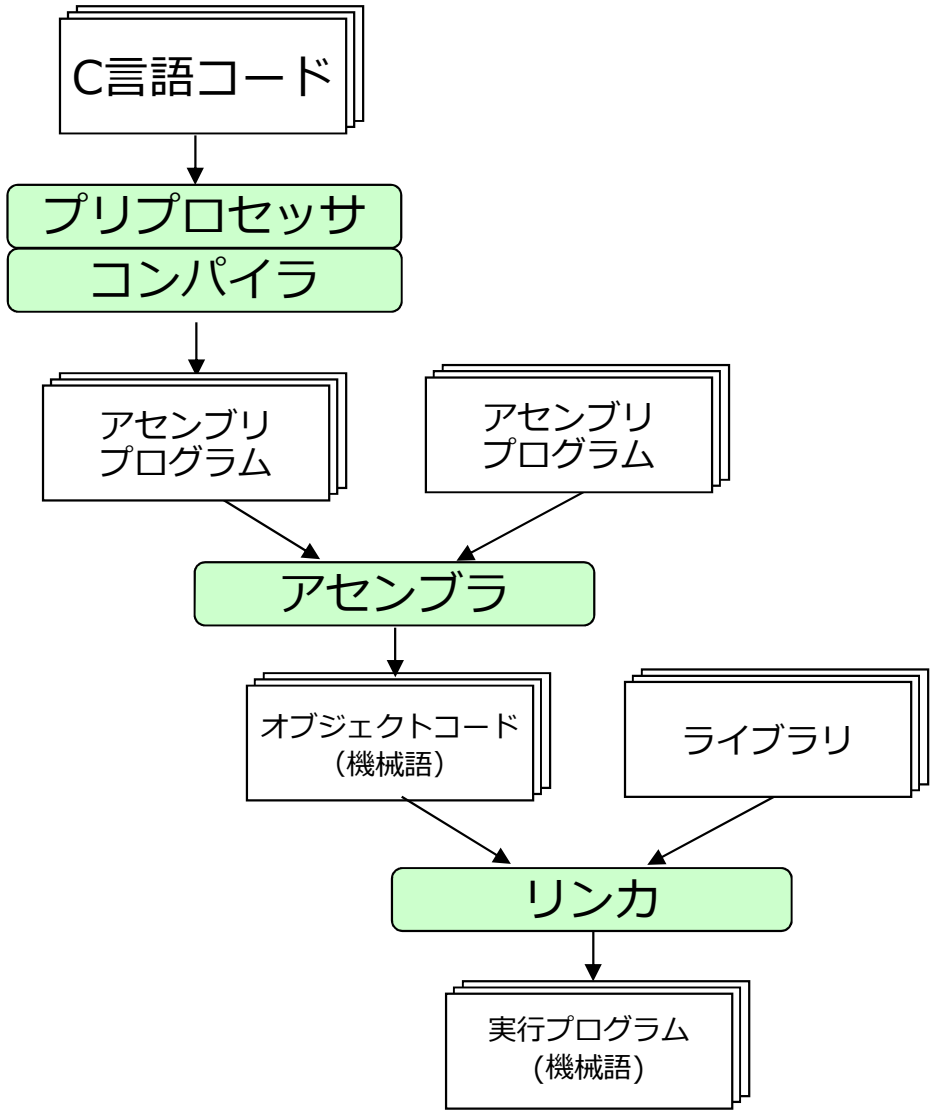
- プロセッサが直接実行可能な命令は機械語
  - 高水準言語は直接実行できない
  - ⇨高水準言語は機械語に変換して実行する必要がある

- 機械語への変換方法
  - コンパイル型
    - 高水準言語のプログラムを事前に機械語に変換して実行する方式
  - インタープリタ型
    - 高水準言語のプログラムを実行時に機械語に変換して実行する方式

プログラム：コンパイル型

- コンパイラ
  - 高水準言語をアセンブリプログラム（アセンブリ言語で記述されたプログラム）へ変換するプログラム
- アセンブラ
  - アセンブリプログラムを機械語のプログラムに変換するプログラム
- 実行プログラム
  - プロセッサで直接実行可能な機械語のプログラム

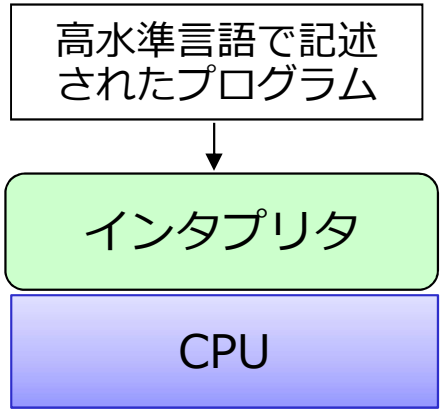
C言語プログラムのコンパイルの流れ



プログラム：インタープリタ型

- インタープリタ
  - 高水準言語のプログラムを機械語に変換して実行するプログラム
- コンパイル型とインタープリタ型の比較

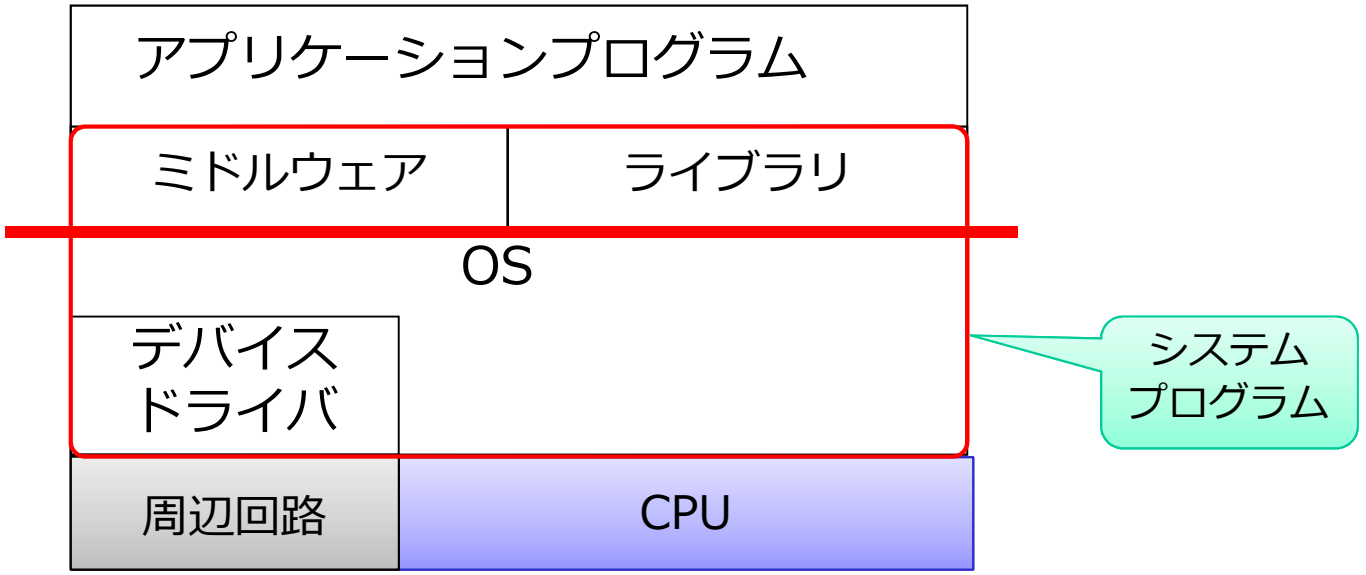
	コンパイル型	インタープリタ型
実行速度	高速 事前に機械語の実行プログラムに変換するため	低速 実行時に機械語に変換するため（様々な高速化技術がある）
汎用性	低い コンパイルした結果は特定の命令セットに依存	高い インタプリタがサポートしている，どの命令セット上でも実行可能
開発効率	低い プログラムの変更後にコンパイルが必要	高い プログラムの変更後はそのまま実行すればよい
秘匿性	高い 人間が読みにくい機械語の実行プログラムを配布	低い 人間が読みやすいプログラムを配布する必要がある



ソフトウェアプラットホーム

ソフトウェアプラットホーム

- アプリケーションプログラム
  - ユーザーの要求に応じて処理を行うプログラム
  - Webブラウザ，LINEアプリ，twitterクライアント
- ソフトウェアプラットホーム
  - アプリケーションソフトウェアの実行を助けるプログラム
  - システムプログラムとも呼ばれる
  - ソフトウェアコンポーネント（モジュール）
    - ソフトウェアを機能により分けた集合



ソフトウェアプラットホーム: 動作例

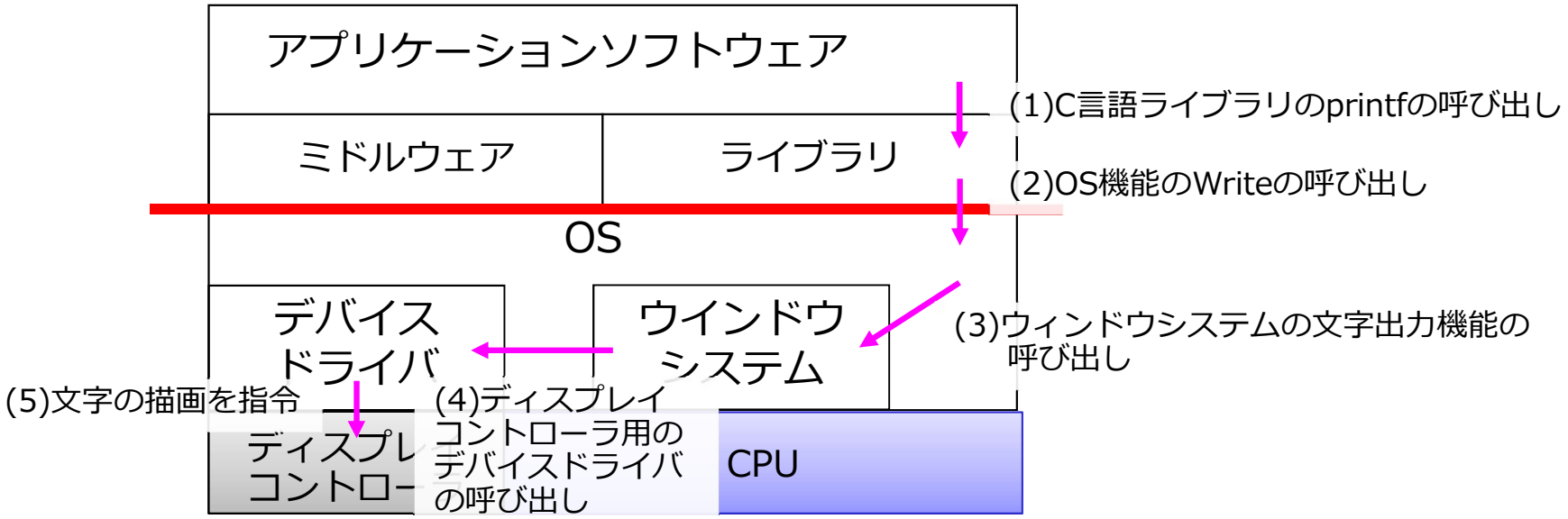
- 文字列を出力するプログラム

```
#include <stdio.h>
int main(void){
    int a = 2;
    printf("a is %d¥n", a);
    return 0;
}
```

コンパイルと実行

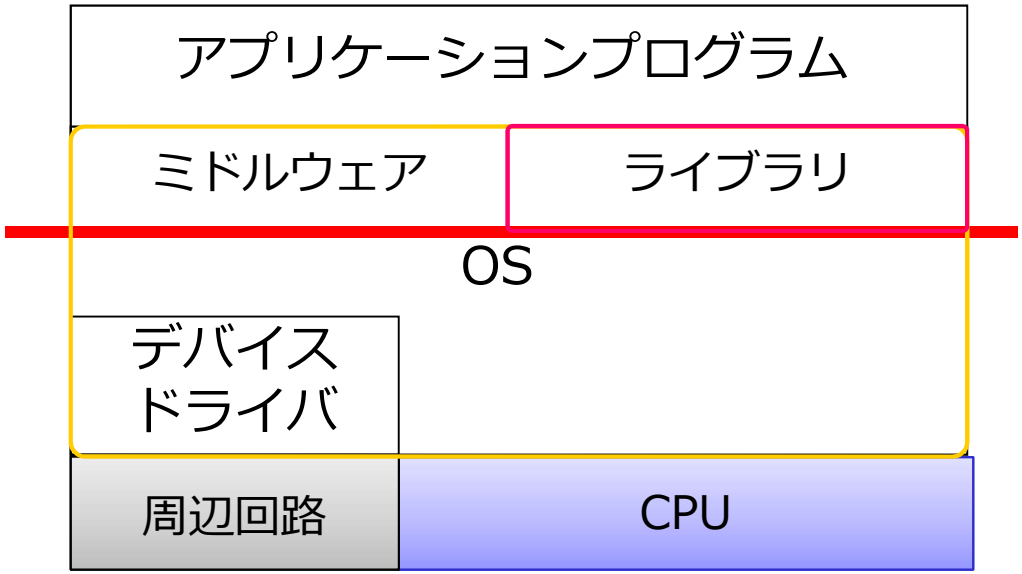
```
~/te
honda@X1G5 ~/te
$ gcc a.c

honda@X1G5 ~/te
$ ./a.exe
a is 2

honda@X1G5 ~/te
$
```

- 文字列が出力されるまでのイメージ



ライブラリ

- アプリケーションソフトウェアで共通的に行う処理を再利用可能な形でひとまとまりにしたもの
  - –プログラム言語で定まっているものもある
  - –例)C言語のライブラリ（標準Cライブラリ）
    - 算術演算
      - ✓三角関数 ：cos・acos，平方根：sqrt
    - 入出力
      - ✓文字出力：putc, 文字入力：getchar



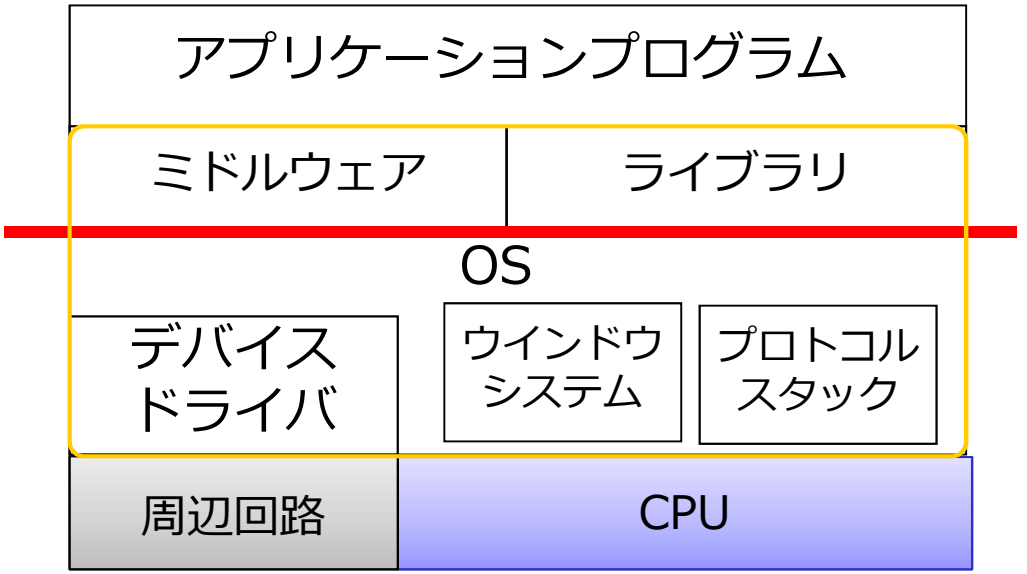
OS（Operating System）

主にハードウェアの仮想化と管理機能を提供するシステムプログラム

- PC用のOS：Windows, Mac OS, Linux
- スマートフォン用のOS：Android, iOS

OSの役割

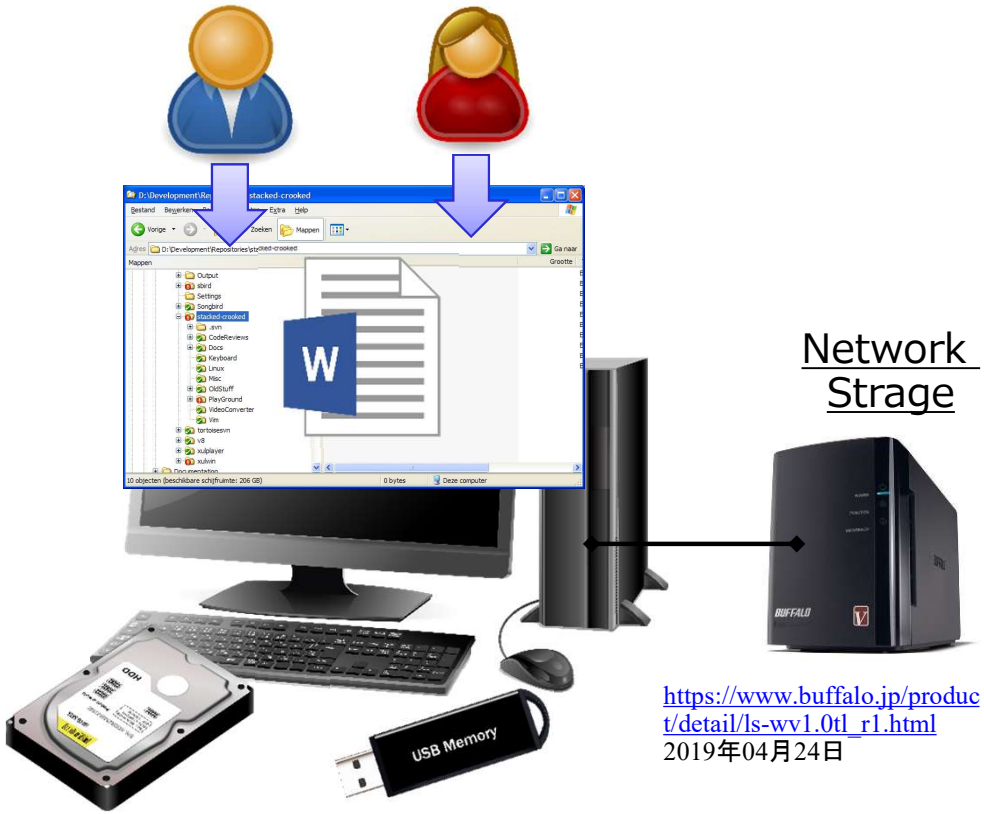
- ハードウェアの仮想化
  - ハードウェアとしてのコンピュータの機能を拡張し使いやすく柔軟なコンピュータを提供すること
  - ハードウェアを抽象化・多重化
- ハードウェア資源の管理機能
  - ユーザにハードウェア資源の共同利用を許し，資源が効率よく使用されるように管理すること



OS：ファイルシステムの例

- ハードウェアの仮想化
  - ディスクというハードウェア資源をファイルやフォルダという形に抽象化・多重化する
  - ユーザーはハードウェアの詳細を知らなくても良い
  - どの様なハードウェアであってもファイルという形で使用出来る

- ハードウェア資源の管理機能
  - 複数のユーザからのアクセスを許す
  - 複数のユーザーが同時にファイルを書き込んだ場合はどちらかの要求のみを処理して，ファイルが壊れないように調停する

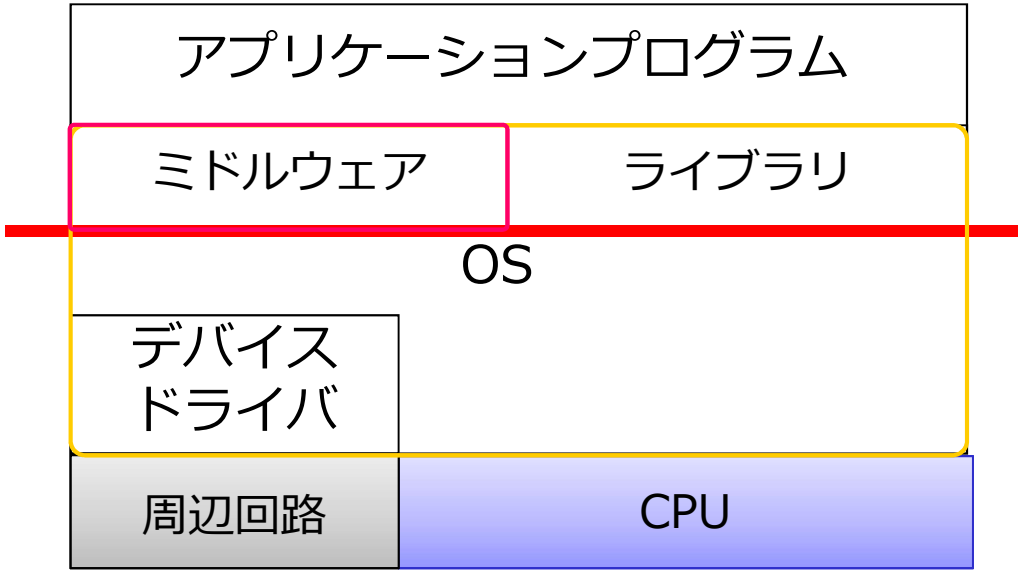


https://www.buffalo.jp/product/detail/ls-wv1.0tl_r1.html
2019年04月24日

ミドルウェア・デバイスドライバ

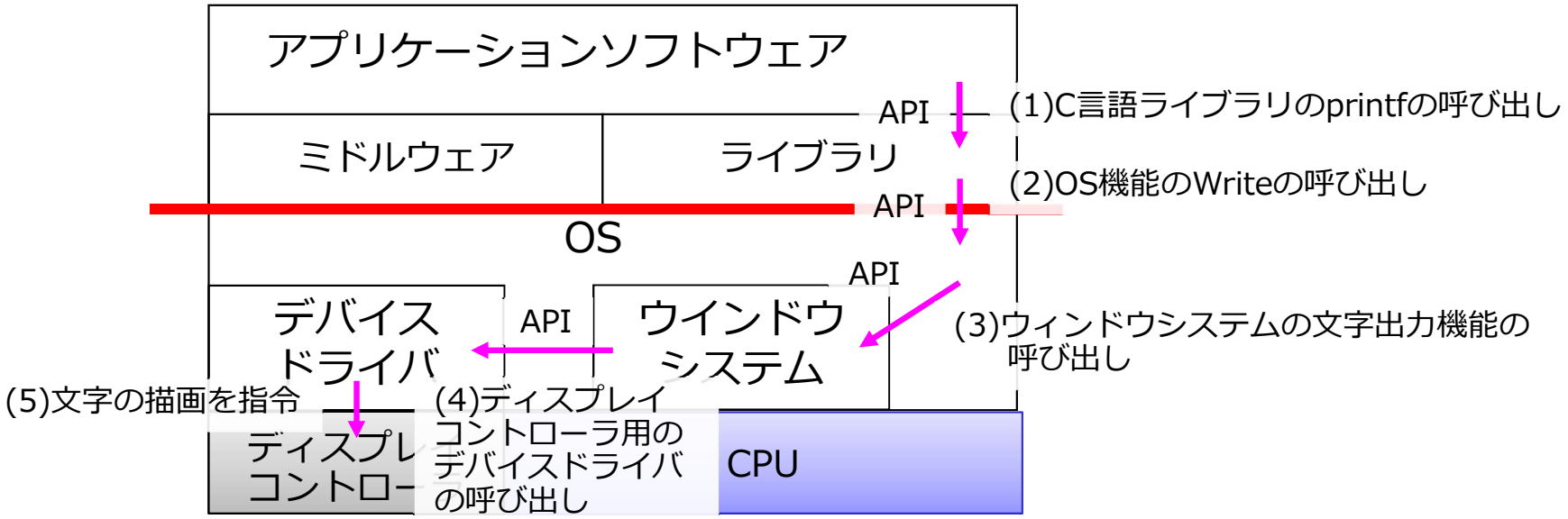
- ミドルウェア
  - アプリケーションプログラムとは別に動作して各アプリケーションプログラムに共通な処理を実現するプログラム
  - ミドルウェアの例
    - データベース管理システム
    - Webサーバー

- デバイスドライバ
  - 周辺回路を制御するプログラム
  - OS内のモジュールとすることで，新しいハードウェアに対応が可能
  - 不具合があるとシステム全体が停止する可能性がある



アプリケーションプログラミングインタフェース（API）

- ソフトウェアコンポーネントが互いにやりとりするのに使用するインタフェースの仕様
- APIを定義して使う理由
  - 呼び出し先のソフトウェアコンポーネントの詳細(設計や実装)を知らなくても使う事ができる
  - 同じAPIをサポートしている異なるコンピュータでアプリケーションプログラムを実行することが可能となる
    - コンパイル型の場合は再度コンパイルする必要がある



まとめ

計算機の仕組みとして，
コンピュータの構成，プログラム言語，
ソフトウェアプラットホームについて説明

- コンピュータの構成
  - プロセッサ，メモリ(ROM/RAM)
  - 命令セット，マイクロアーキテクチャ
  - マルチプロセッサ，周辺回路
- プログラム言語
  - コンパイル型，インタープリタ型
- ソフトウェアプラットホーム
  - ライブラリ，OS，デバイスドライバ，ミドルウェア，API

参考文献

- 川合　慧［編］：情報，東京大学出版会，2006.
- 築山他：ビジュアルに学ぶディジタル回路設計, コロナ社, 2010.
- Wikipedia
- ARM Webサイト
- Intel Webサイト

レポート課題

- 以下の設問に答えよ．A4版の紙に回答し，次回（6月29日）のインフォマティクス１の時間に提出.
- 学籍番号と名前を忘れないように．質問等はhonda@ertl.jp（本田）まで.

1. 自分が使用しているスマートフォンないしはPCの仕様について調査せよ. 持っていない場合は適当なスマートフォンを選んで調査せよ
   - ハードウェア：コア数，動作周波数，ROM/RAMのサイズ，命令セット
   - ソフトウェア：OS，アプリケーションの開発言語
2. 次のプログラムをコンパイル型とインタープリタ型のどちらで作成するべきかその理由と共に答えよ.
   - 多くの計算が必要な科学計算プログラム
   - WindowsとMacで使用する，履修状況から卒業要件を満たしているか判定するプログラム
   - データを暗号化するプログラム
3. プロセッサの命令セットやソフトウェアコンポーネントのAPIはその機能が追加されることはあっても，逆に機能が削除されることはほとんどない．その理由について説明せよ.