計算機アーキテクチャ

システム・オン・チップ (1)
組込みプロセッサ

一色 剛

工学院情報通信系

isshiki@ict.e.titech.ac.jp

講義概要(3回分)

- （第1回）組込みプロセッサ：マイクロコントローラ（マイコン）
  - Intel 8051 8ビットマイコン命令セット、タイマー、割込み制御
  - Internet-of-Things (IoT)への課題

  → 組込みシステムにおけるマイクロコントローラの役割・課題を理解
- （第2回）車載用SoC
  - ECU (electrical Control Unit)、システム構成、ADAS技術
  - 組込みソフトウェア階層、マルチタスク処理

  → 車載システム特有のシステム構成・技術要件、今後の動向を理解
- （第3回）携帯電話・スマートフォン用SoC
  - カスタムHW（画像信号処理、コーデック）
  - GPUアーキテクチャ、ARMプロセッサ

  → 組込み分野で最も技術革新が激しいスマホを支える技術を理解

本日の小テスト

（配布用紙に学籍番号・氏名を記入し、講義終了後に回収します）

下記設問2問以上の回答と講義の感想（自由記述）

1. 組込みシステムの構成要素、機能・用途、要件を簡単に述べよ。
2. 2013年当時で台数ベースシェアが多い順に並べよ。
   - A) 4/8ビットマイコン
   - B) 16ビットマイコン
   - C) 32ビットマイコン
3. Intel 8051マイコンのデータ幅、アドレス幅、ACC/SP/PCの各レジスタ幅を答えよ。また、ADDのような演算命令の第2オペランドの4つのアドレス形式と、それぞれの命令長（バイト数）を答えよ。
4. IoT時代において従来型の組込みシステムが抱える課題を述べよ。

組込みシステムとシステム・オン・チップ

■ 組込みシステム

- 特定の用途に特化した計算機システム
- 構成要素：プロセッサ、メモリ、入出力インターフェース、専用処理回路（カスタムハードウェア）、ソフトウェア

■ システム・オン・チップ：System-on-Chip (SoC)

- システムの構成部品のほとんどを1チップに集積

Qualcomm Snapdragon 810 → Androidスマホ
チップ面積の67%がカスタムHWで占める
（GPS, Display, Modem, USB, Camera ISP, audio/video codec）

http://www.tomshardware.com/reviews/snapdragon-810-benchmarks,4053-2.html

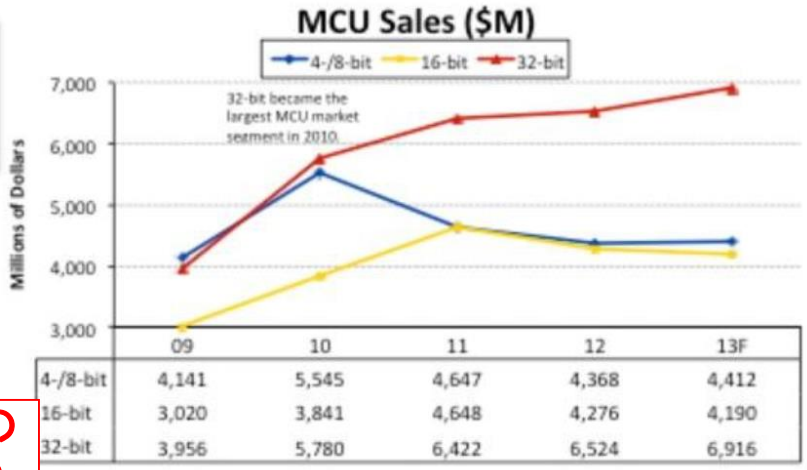
組込みシステムの特性

- **機能・用途 : 機器制御**
  - 入力処理 : 環境信号（センサー、カメラ）、人間操作（キー、タッチパネル、音声）
  - 出力処理 : 情報出力（コンソール、LCD、音声）、機器駆動出力（モータ、照明等）
  - 通信処理 : シリアルポート（UART, SPI）、ネットワーク（有線・無線）
  - 計算処理 : 機器ごとに異なる（プログラム制御）
- **要件**
  - 低コスト、省スペース
  - 厳しい資源制約（メモリ容量、演算性能、消費電力等）
  - 厳しい動作環境（温度、振動等）
  - 短い開発期間（組込み市場の速い動向変化）
  - リアルタイム性、高信頼性
  - 多数の入出力処理要求

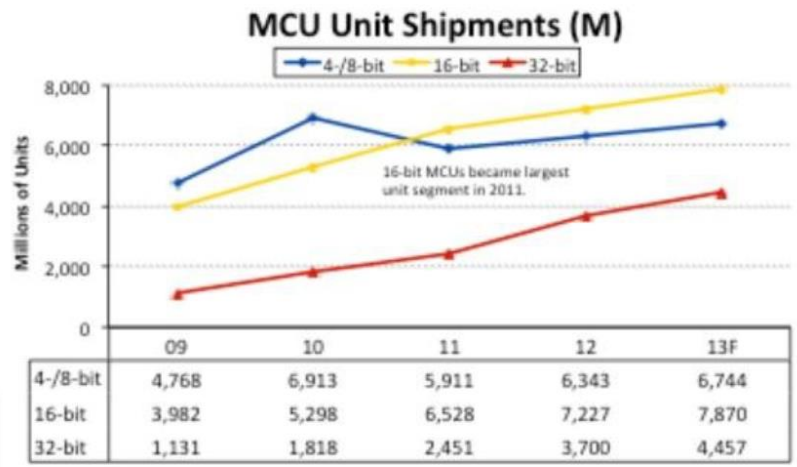
組込みプロセッサ（マイクロコントローラ）

- **組込みシステム用CPU（マイクロコントローラ：MCU）**
  - 主機能：外部機器の制御
  - ビット幅：4, 8, 16, 32
- **MCU市場規模 (2013)**
  - 総額：152億ドル
  - 出荷数：175億個
- **MCU以外のマイクロプロセッサ市場規模 (2013)**
  - 総額：610億ドル
  - PC/Servers (56%)
  - Cellphone/Tablet (32%)

4/8-bitマイコンの需要はまだある



	09	10	11	12	13F
4-/8-bit	4,141	5,545	4,647	4,368	4,412
16-bit	3,020	3,841	4,648	4,276	4,190
32-bit	3,956	5,780	6,422	6,524	6,916



	09	10	11	12	13F
4-/8-bit	4,768	6,913	5,911	6,343	6,744
16-bit	3,982	5,298	6,528	7,227	7,870
32-bit	1,131	1,818	2,451	3,700	4,457

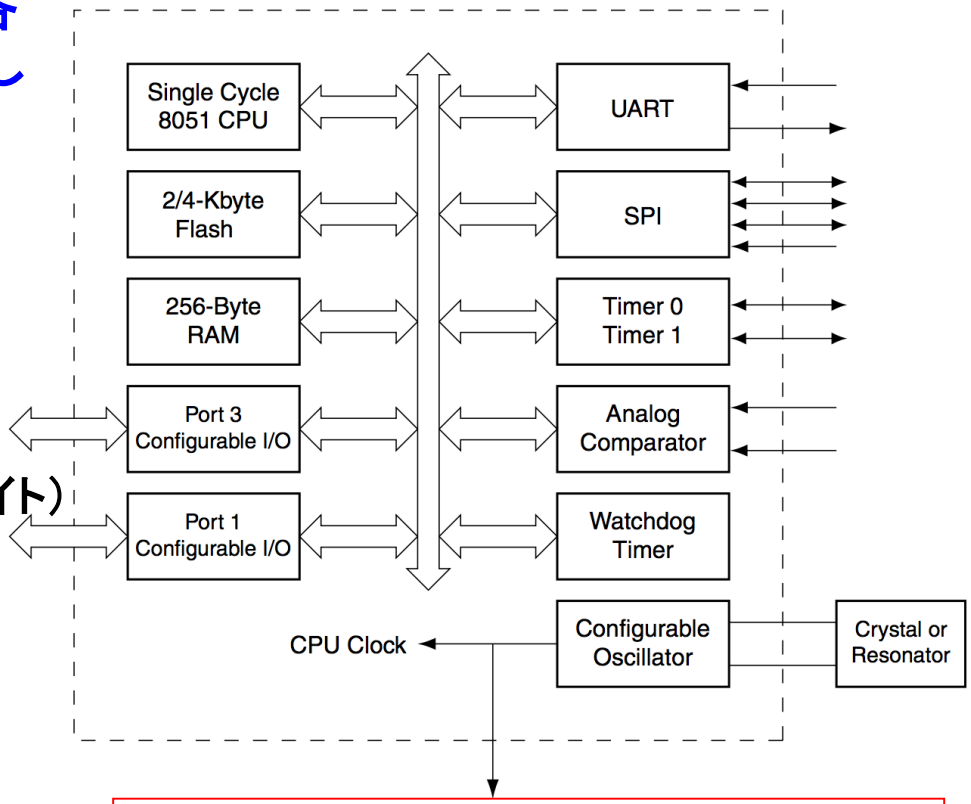
Source: IC Insights

Source: http://www.icinsights.com/news/bulletins/Tablet-And-Cellphone-Processors-Offset-PC-MPU-Weakness

Source: http://www.eetimes.com/document.asp?doc_id=1280803

Intel 8051 (1980～)

- 現在でも互換チップ・派生品が各社から提供されている最も普及した8ビットマイクロコントローラ
- CPU core:
  - ALU（算術論理、乗算、除算）
  - データ幅 : 8 bits
- メモリ : ハーバード型
  - プログラム・データ分離メモリ
  - アドレス幅 : 16 bits（最大64Kバイト）
- 割込み機構
- 周辺回路
  - タイマー x 3
  - IOポート (8-bit, 最大4IO)
  - シリアルポート (UART, SPI)



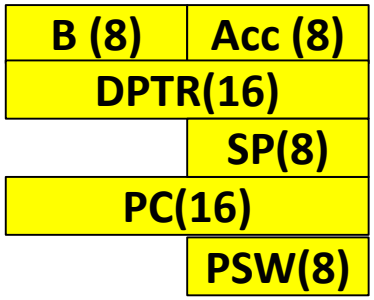
Source: http://www.atmel.com/ja/jp/Images/doc3547.pdf

Source: cc.cqpub.co.jp/lib/system/document_download/1090/1/

Intel 8051 (1980～)

■ レジスタ:

- Acc : 8-bit アキュムレータ
- B : 乗算・除算で使用
- DPTR : 16-bit（MOV命令で使用）
- SP : 8-bitスタックポインタ
- PC : 16-bitプログラムカウンタ
- PSW : 8-bit状態レジスタ（carry, overflow, parity, reg-bank sel[2]）



■ 内部データメモリ（128/256バイト）:

- 下位32バイトは「レジスタ」としてアクセス可能 → 4バンク x 8ワード（バンクはPSWで切替）
- スタックメモリ領域としても使用

■ 外部メモリ:

・データメモリ : 最大64Kバイト

・プログラムメモリ : 最大64Kバイト

■ IO, TIMER:

・データメモリへの書込み・読出しと同じ命令体系

Source: http://www.st.rim.or.jp/~nkomatsu/intel8bit/i8051.html

Intel 8051 (1980～)

- **Acc オペランド:**
  - 第1演算オペランド、出力オペランド
- **第2オペランド:**
  - レジスタ（内部メモリ下位32バイト）→ 8ワード位置を命令中の3ビットで指定
  - 直接アドレス（内部メモリ）→ 128/256ワード位置を命令中8ビットで指定
  - 間接アドレス → R0/R1のいずれか（命令中1ビット指定）の値をアドレスとしてメモリにアクセスする
  - 即値 → 1バイト即値を命令中で指定

ADD命令形式

- ADD A, Rn : A ← A + Rn

(n = 0～7 : レジスタ番号)

- *ADD A, adr : A ← A + M[adr]

(aa : 8-bit内部メモリアドレス)

- ADD A, @Ri : A ← A + M[Ri]

(Ri : R0 or R1)

- *ADD A, #data : A ← A + #data

(#data : 8-bit即値データ)

* : 2バイト命令（他は1バイト命令）

→同形式の他命令 : ADDC, SUBB, INC, DEC, AND, OR, XOR

Source: http://www.st.rim.or.jp/~nkomatsu/intel8bit/i8051.html

Intel 8051 (1980～)

■ MOV命令形式

- MOV A, Rn : A ← Rn
- MOV A, adr : A ← M[adr]
- MOV A, @Ri : A ← M[Ri]
- MOV A, #data : A ← #data
- MOV Rn, A : Rn ← A
- MOV Rn, adr : Rn ← M[adr]
- MOV Rn, #data : Rn ← #data
- MOV adr, A : M[adr] ← A
- MOV adr, Rn : M[adr] ← Rn
- MOV adr, sadr : M[adr] ← M[sadr]
- MOV adr, @Ri : M[adr] ← M[Ri]
- MOV adr, #data : M[adr] ← #data
- MOV @Ri, A : M[Ri] ← A
- MOV @Ri, adr : M[Ri] ← M[adr]
- MOV @Ri, #data : M[Ri] ← data

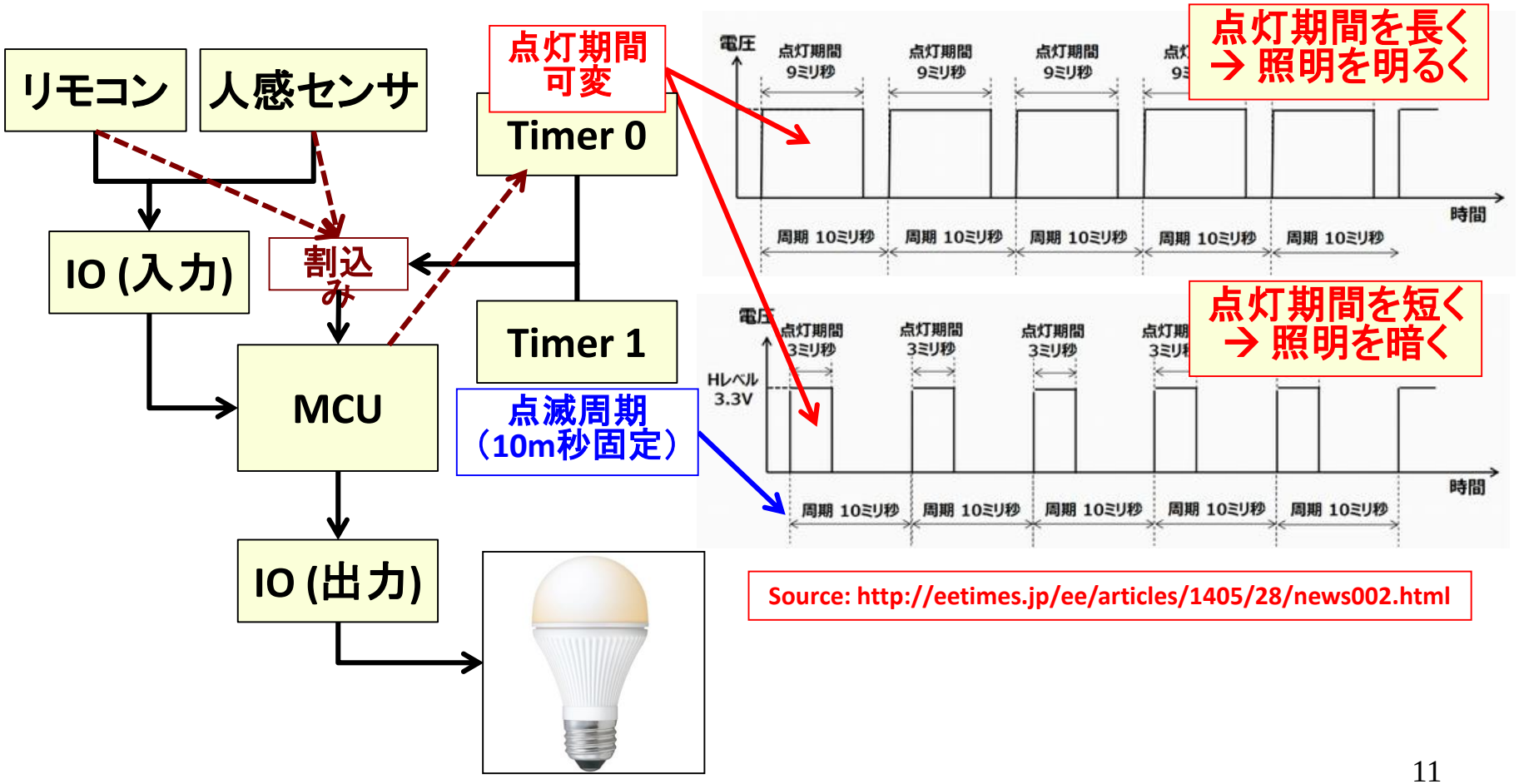
その他のmov系命令

- 外部メモリとの転送 : MOVC, MOVX
- スタック操作 : PUSH, POP
- データ交換 : XCH

Source: http://www.st.rim.or.jp/~nkomatsu/intel8bit/i8051.html

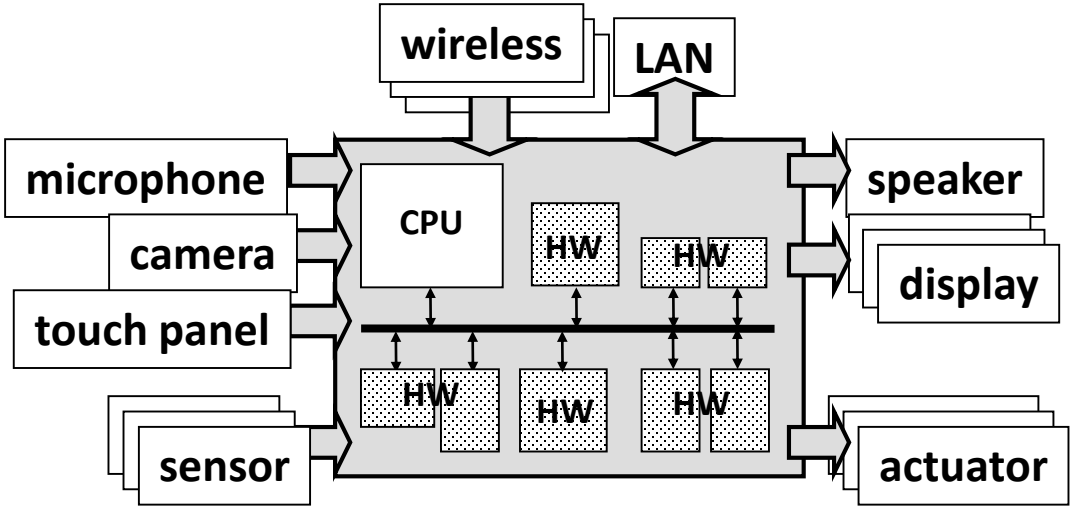
マイクロコントローラ動作例：照明制御

■ PWM (Pulse Width Modulation)による照度調整



Source: http://eetimes.jp/ee/articles/1405/28/news002.html

組込みシステムにおける ハードウェア制御方法



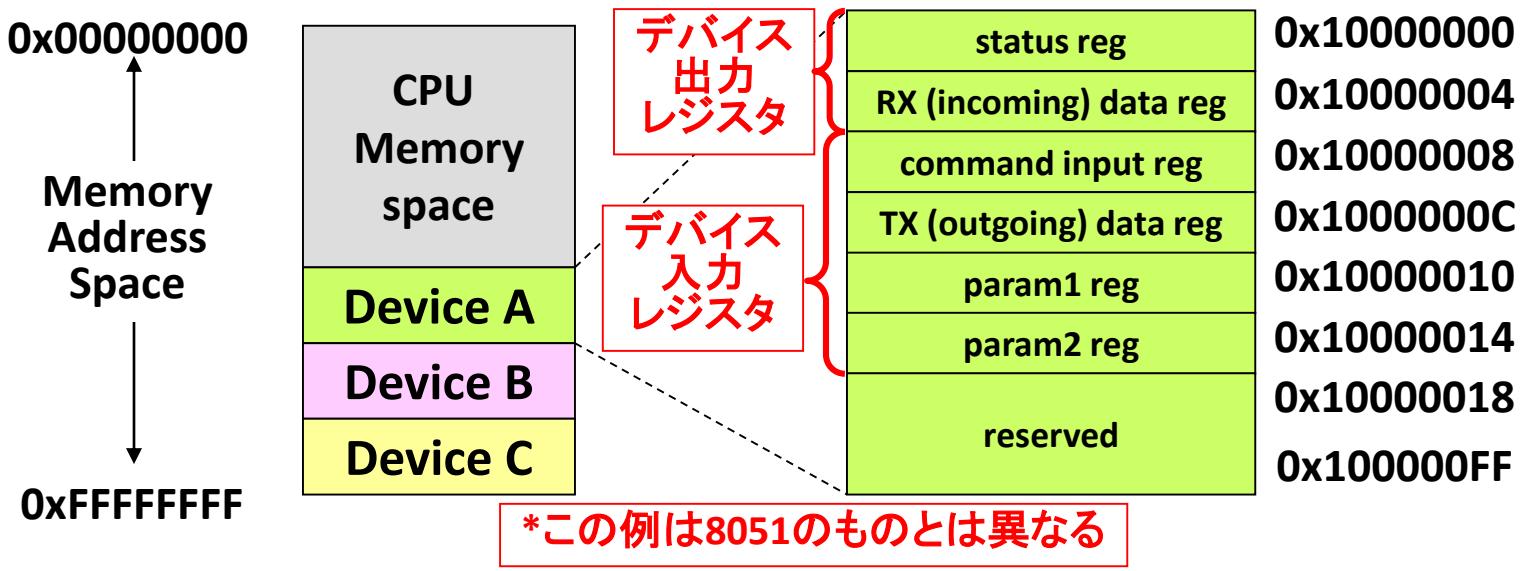
- 2種類の「ハードウェア」
  - 特定の計算処理を行う専用回路（タイマー、画像・音声処理、等）
  - 外部機器と通信を行うためのインターフェース回路
- 「ソフトウェア」はどのようにしてこれら「ハードウェア」と会話するか？

組込みシステムにおける ハードウェア制御方法

- 「ハードウェア」への基本命令 : “Write” “Read”
  - Write: ハードウェアへデータを書き込む
    - コマンド:ハードウェアに対し何らかの指令を出す (ハードウェア上のパラメータ設定、特定処理実行開始、等)
    - データ送信 : 外部メモリへの書き込み、シリアルバス出力等
  - Read: ハードウェアからデータを読み出す
    - ステータス（状態）: ハードウェア状態 (“ready”/“busy”等)
    - データ受信 : 外部メモリから読み出し、シリアルバス入力等
- どこにRead/Writeすべきか?
  - ハードウェア専用メモリ空間 : Memory-mapped device → 8051の方式を踏襲

Memory-Mapped Devices

- 固有アドレス空間を割り当てられたハードウェア・デバイス
  - ハードウェアへのRead/Writeはデバイス固有アドレス空間に対して行う
- レジスタインターフェース
  - デバイス固有アドレス空間内に個別レジスタアドレスを配置
  - 各レジスタ : 入力専用、出力専用、入出力両用



ソフトウェアからのデバイスアクセス

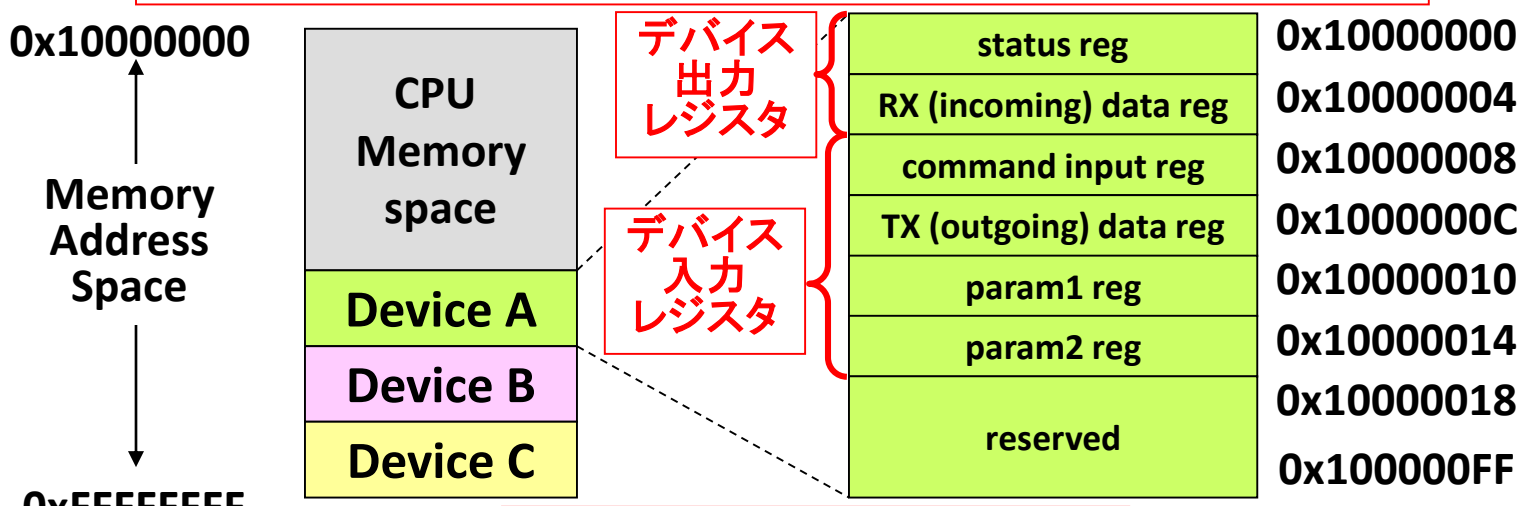
- 各レジスタアドレスを直接アクセスする記述をすれば良い
  - "status reg"の読み出し：デバイス状態の検知

```
int status = *(volatile int *)(0x10000000);
```

  - "TX data reg"への書き込み ： データ送信

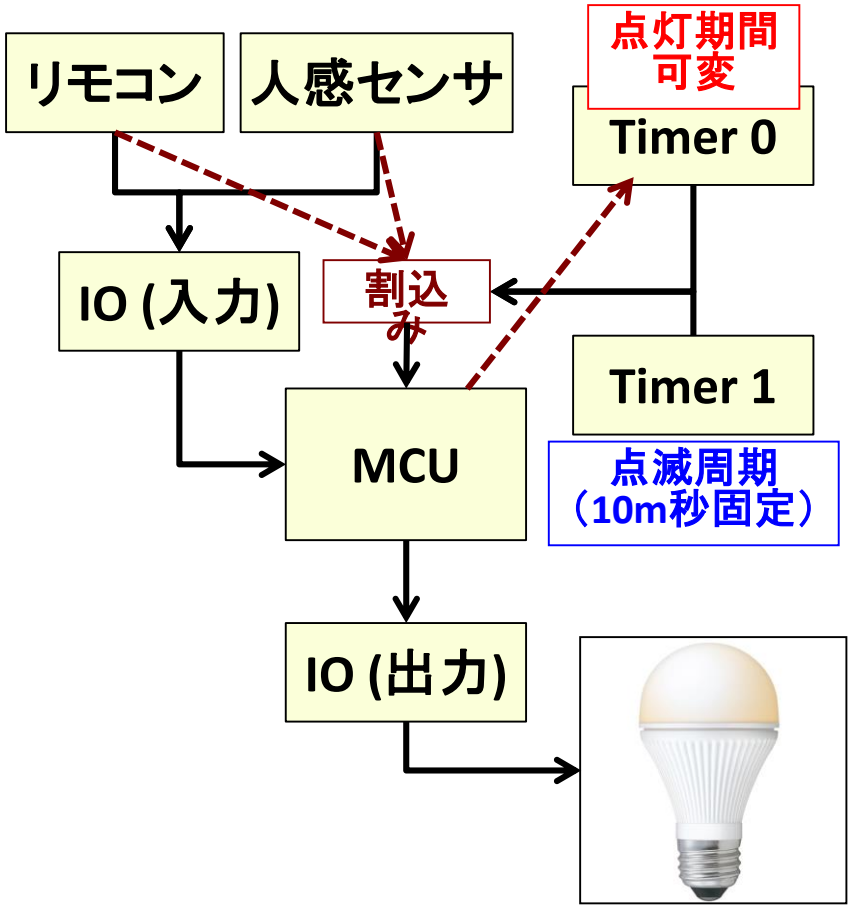
```
*(volatile int *)(0x1000000C) = output_data;
```

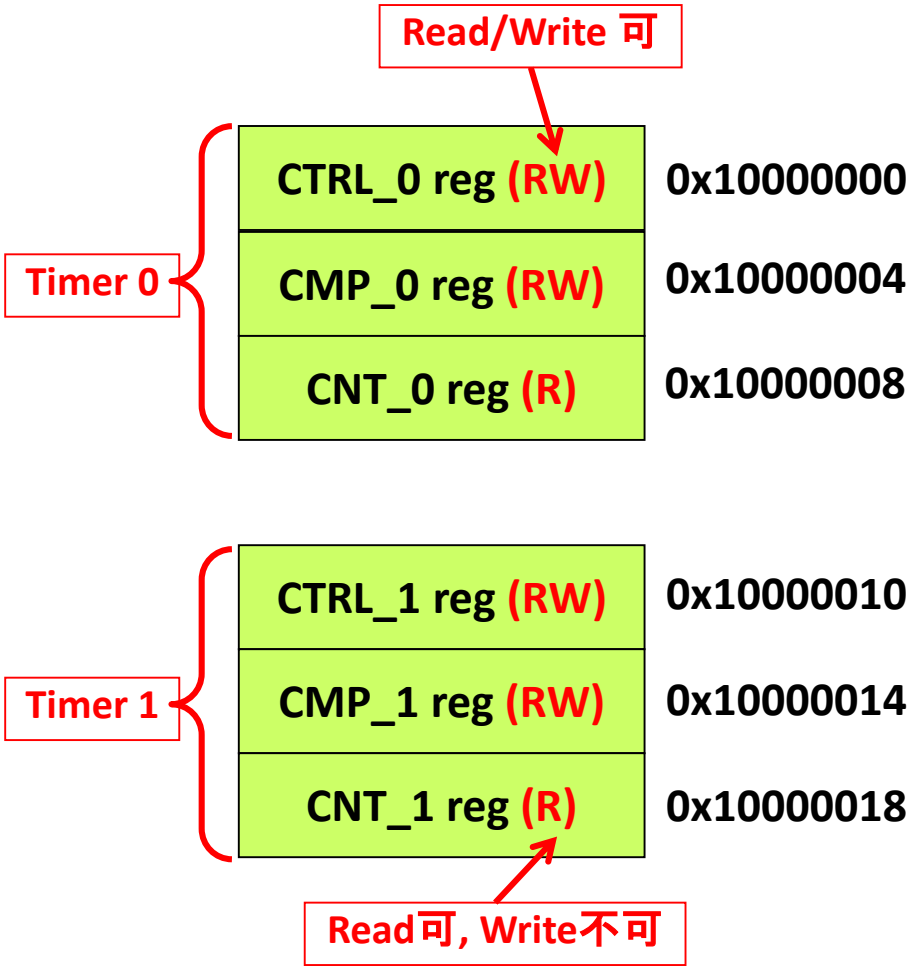
0x10000004番地を(volatile int)型データのアドレス（ポインタ）と見なし、このアドレスにデータを「書き込む」



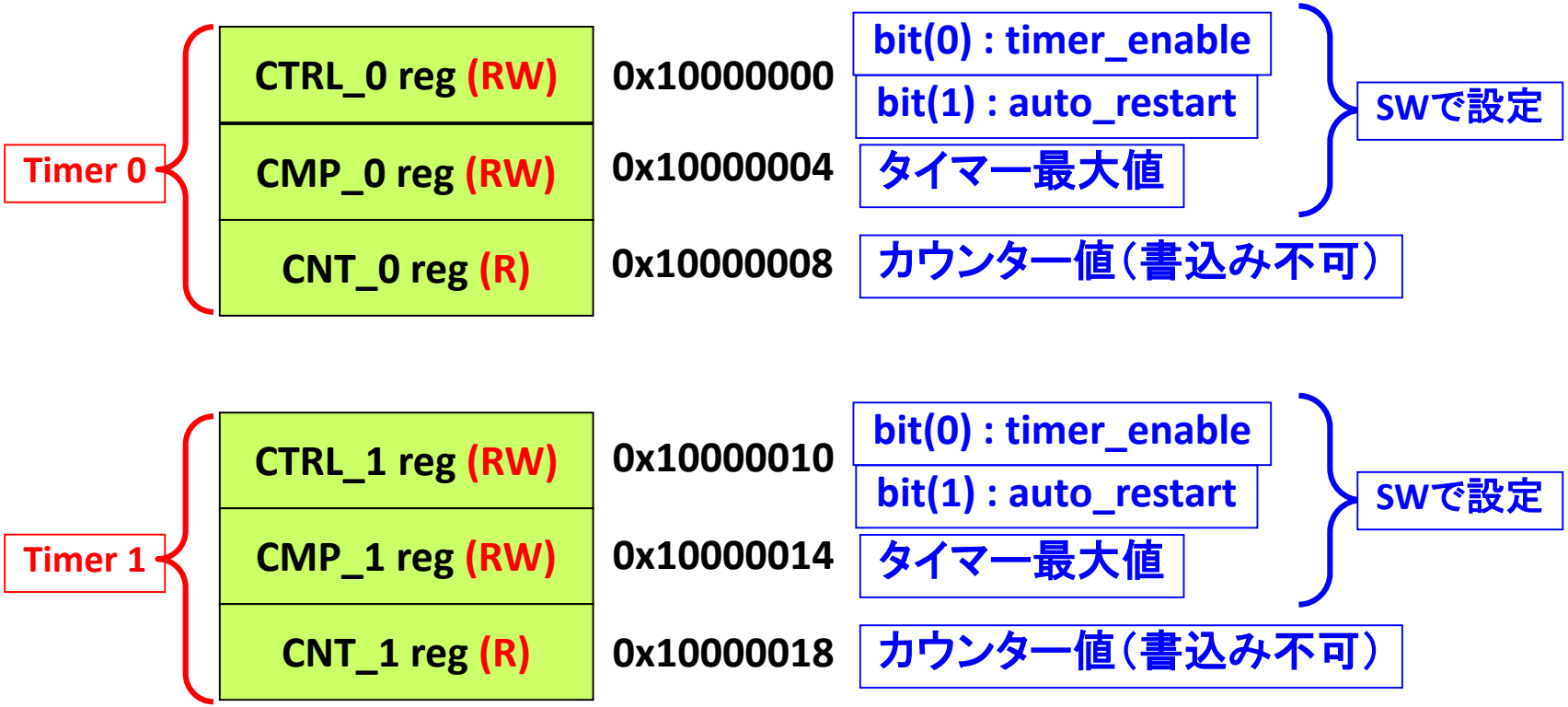
*この例は8051のものとは異なる

タイマーのレジスタインターフェースの例





タイマの各レジスタの機能



上記は標準的なタイマー制御方式
（色んなバリエーションはあり得る）

タイマーのHW論理動作（毎クロック）

CTRL_0/1 reg (RW)

bit(0) : timer_enable

bit(1) : auto_restart

```
Update_timer (n) {  /// n = 0, 1
  if (CTRL_reg[n] & 1) {
    /// timer_enable
    if (CNT_reg[n] == CMP_reg[n]) {
      /// timer limit reached!!
      if (CTRL_reg[n] & 2) {
        /// auto-restart mode
        CNT_reg[n] = 1;  /// restart from 1
      }
      else {
        /// NOT auto-restart mode
        CTRL_reg[n] = 0; /// timer_disabled
      }
      /// CPUへ割込み通知!!
      Generate_Timer_Interrupt(n);
    }
    else { CNT_reg[n] ++; } /// count up
  }
}
```

TIMERカウンタが指定最大値に到達

auto-restartモードの場合、カウンタ値を1に戻し、タイマーを継続する（周期タイマー）

auto-restartモードでない場合、タイマーを停止する（単発タイマー）

TIMERカウンタが最大値に到達した時にCPUへ割込みを通知する

最大値未満の場合、TIMERカウンタをインクリメント

タイマーデバイス制御プログラム

```
void start_timer_1 (int limit_1) {
  /// 10m秒になるように limit_1 を設定
  *(volatile int *) TIMER_CMP_1 = limit_1;
  /// auto_restart = 1, timer_enabled = 1;
  /// 周期タイマー
  *(volatile int *) TIMER_CTRL_1 = 3;
}
```

割込

システム
起動時のみ

```
void interrupt_timer_1 () {
  /// 10m秒周期の開始
  Set_LED_Output (1);
  /// start TIMER_1
  start_timer_0 (brightness);
}
```

割込ハンドラ 1

10m秒毎
周期的に
発火

Timer_1の割込みハンドラから
周期的に呼ばれる

```
void start_timer_0 (int limit_0) {
  /// limit_0で明るさを調整
  *(volatile int *) TIMER_CMP_0 = limit_0;
  /// auto_restart = 0, timer_enabled = 1;
  /// 単発タイマー
  *(volatile int *) TIMER_CTRL_1 = 1;
}
```

割込

点灯期間
終了後

```
void interrupt_timer_0 () {
  ///「点灯期間」終了
  Set_LED_Output (0);
}
```

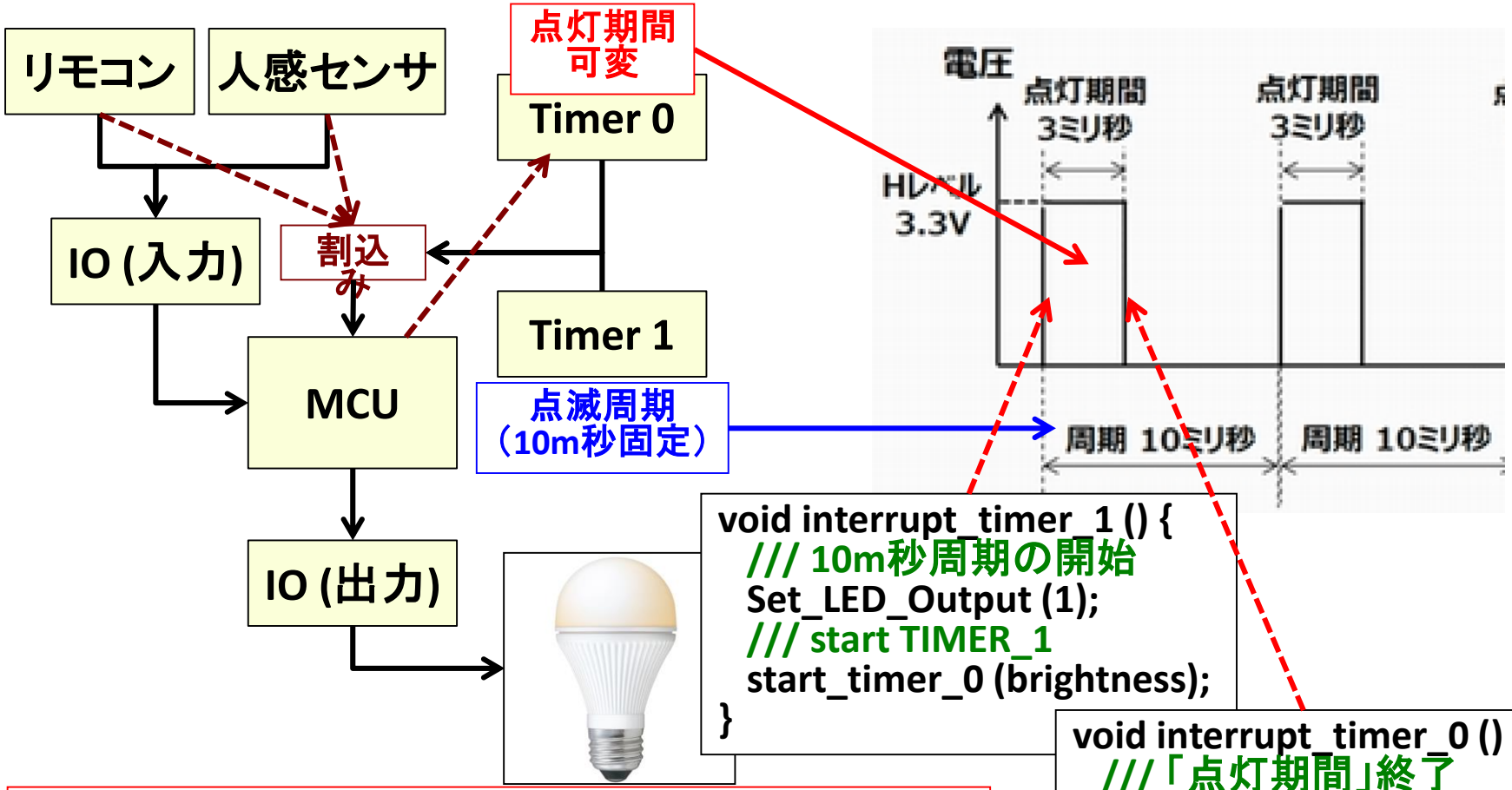
割込ハンドラ 0

CTRL_0/1 reg (RW)

bit(0) : timer_enable

bit(1) : auto_restart

PWMによる照度調整動作



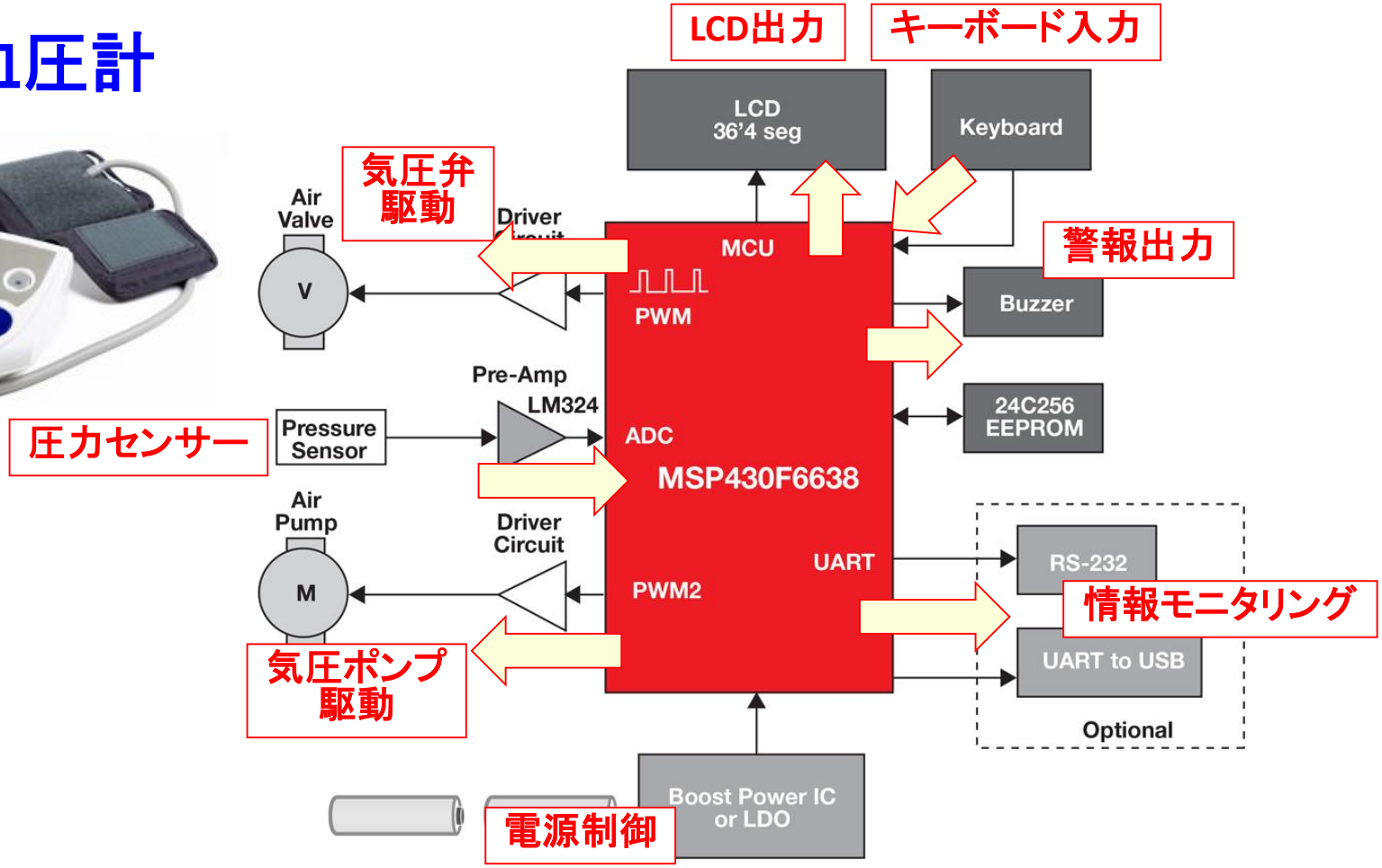

```
void interrupt_timer_1 () {
  /// 10m秒周期の開始
  Set_LED_Output (1);
  /// start TIMER_1
  start_timer_0 (brightness);
}
```

```
void interrupt_timer_0 () {
  ///「点灯期間」終了
  Set_LED_Output (0);
}
```

単純な機器制御では問題ないが、多数の入出力に最適応答するような複雑なシステムでは、単純な割込み制御だけではうまく実装ができない → OSが必要

マイクロコントローラ応用例 (1)

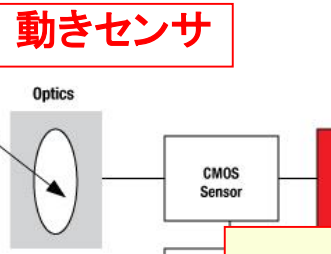
■ 血圧計



Source: http://www.tij.co.jp/lsds/ti_ja/microcontrollers-16-bit-32-bit/msp/applications.page

マイクロコントローラ応用例 (2)

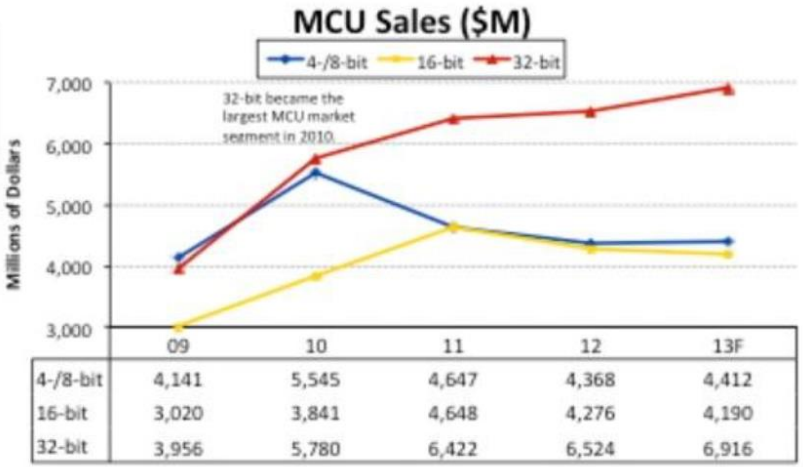
■ 占有検出・モーション検出



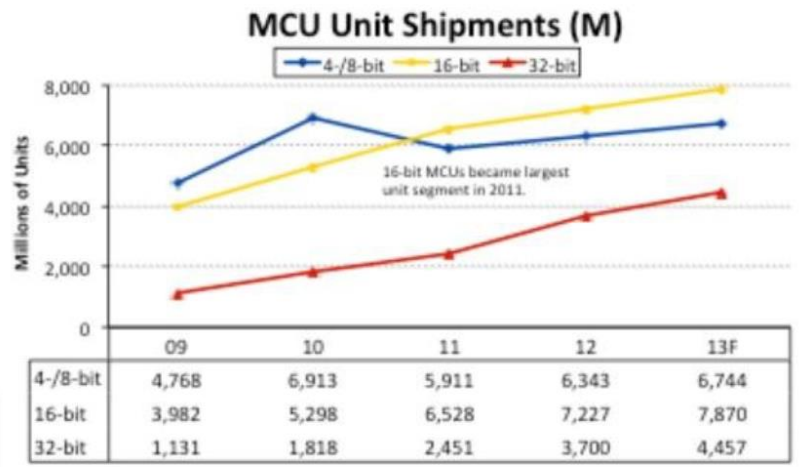
Source: http://www.tij.co.jp/lsds/ti_ja/microcontrollers-16-bit-32-bit/msp/applications.page

「今までの」組込みシステム

- **4/8ビットマイコンの出荷数は32ビットよりも依然として多い(2013年現在)**
  - シェアは下がっているが出荷数ベースでは微増
- **これら「従来型」組込みシステム→大半がstand-alone型**
  - 他システムと繋がることさえ想定されていないものも多い
  - 計算能力の進化を止めたままでも付加価値を生み出し続けている



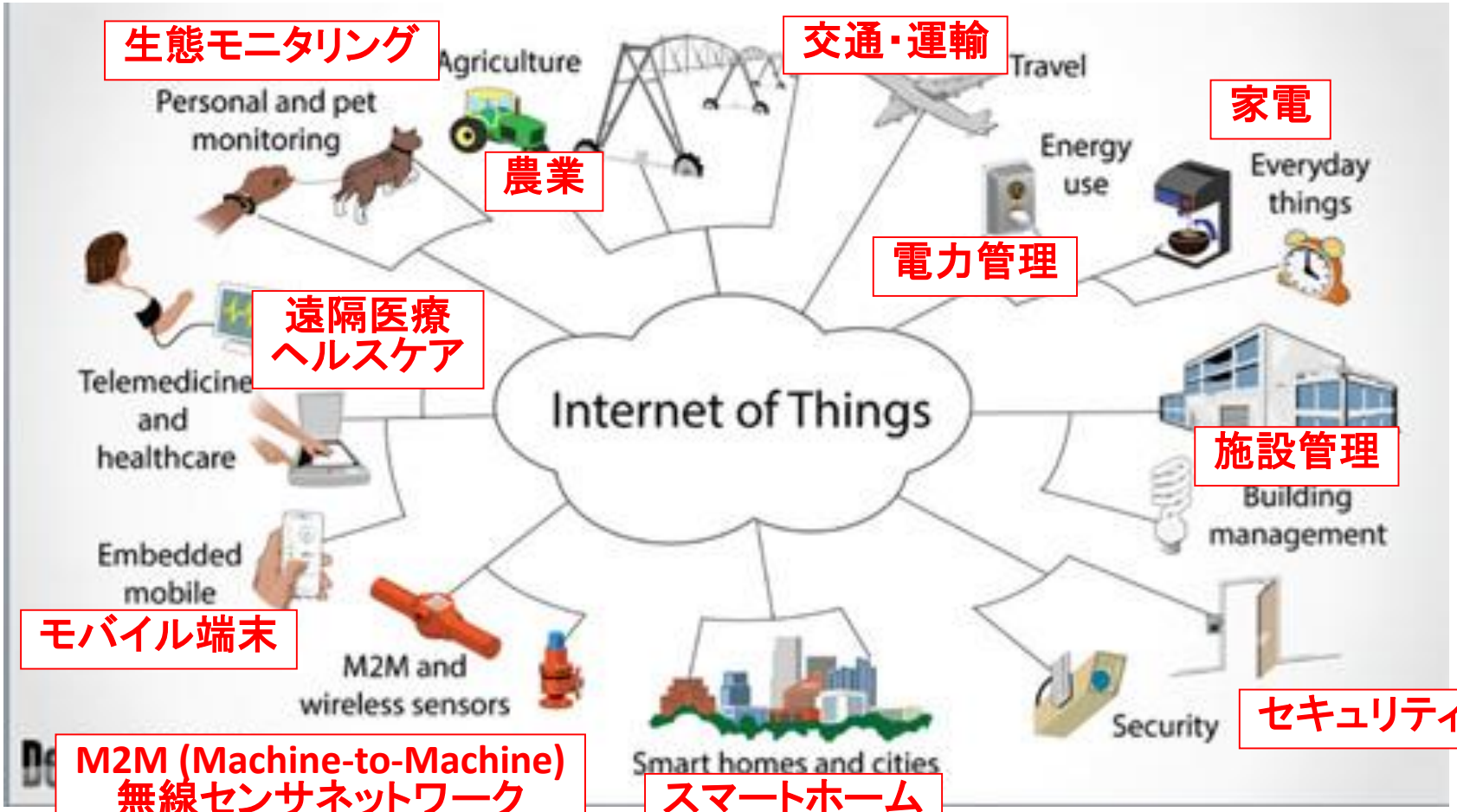
	09	10	11	12	13F
4-/8-bit	4,141	5,545	4,647	4,368	4,412
16-bit	3,020	3,841	4,648	4,276	4,190
32-bit	3,956	5,780	6,422	6,524	6,916



	09	10	11	12	13F
4-/8-bit	4,768	6,913	5,911	6,343	6,744
16-bit	3,982	5,298	6,528	7,227	7,870
32-bit	1,131	1,818	2,451	3,700	4,457

Source: IC Insights

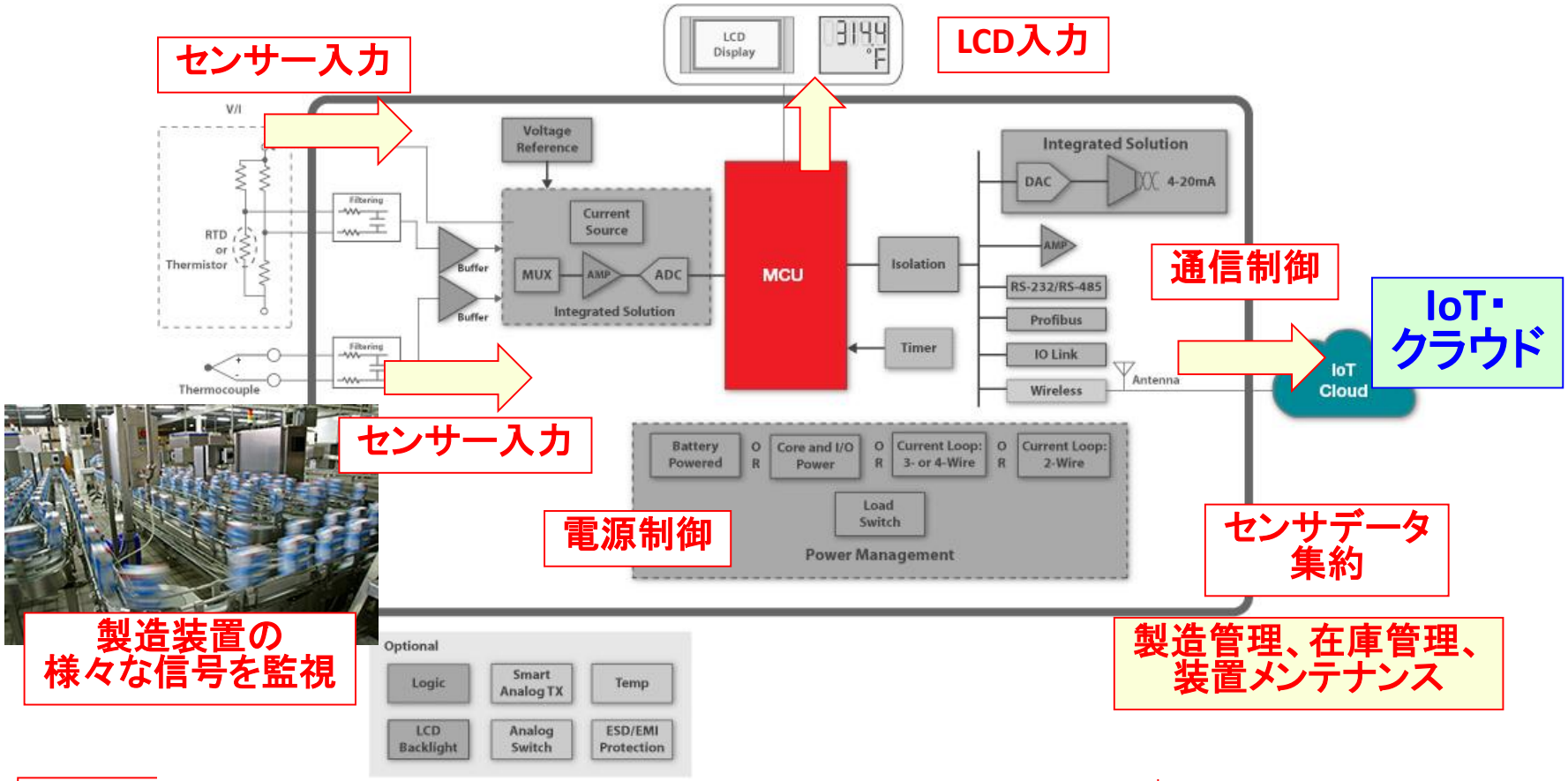
「これから」の組込みシステム→全て繋がる



Source: http://www.designworldonline.com/tips-on-designing-for-the-internet-of-things

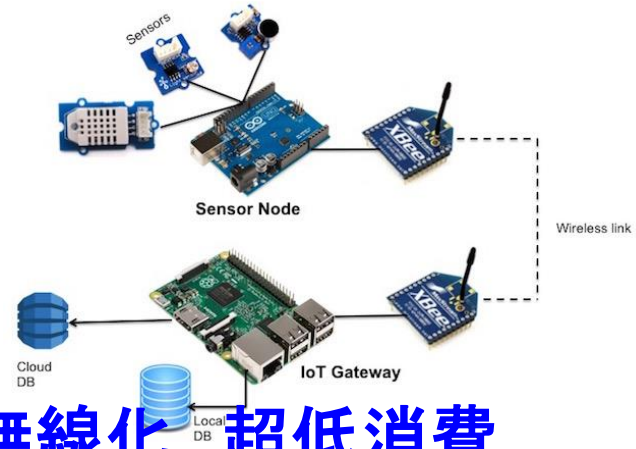
マイクロコントローラ応用例 (3)

■ ファクトリ・オートメーション



Source: http://www.tij.co.jp/lsds/ti_ja/microcontrollers-16-bit-32-bit/msp/applications.page

IoTへの課題 (1)



Source: https://thenewstack.io/tutorial-prototyping-a-sensor-node-and-iot-gateway-with-arduino-and-raspberry-pi-part-1/

■ センサノード：小型化・低コスト化、無線化、超低消費電力、知的信号処理・環境認識

- 小型化・低コスト化：センサデバイスへの無線機能集積化 → 現状は、既製品（センサ、マイコン、無線デバイス）のボード集積による試作が中心（上図：Rasberry-piセンサノード）
- 超低消費電力：センシング・データ処理を環境発電（振動、太陽光）で駆動可能なレベルの超低消費電力回路
- 知的信号処理・環境認識：センシングデータの垂れ流し（クラウド一括処理）ではなく、知的信号処理による環境認識、特徴抽出までを行うことで、爆発的に増加するIoT通信量を激減

→センサ技術 + 無線技術 + 信号処理回路技術の集大成が必要

IoTへの課題 (2)

Source: http://eetimes.jp/ee/articles/1506/22/news069_2.html

■ ネットワーク：センサノード・ゲートウェイ・クラウド間

- 多様なセンサノードやゲートウェイ機器を相互接続するための通信制御方式の確立
- セキュリティ：実装置に直接接続するセンサ・アクチュエータを攻撃者から保護するための強靭なセキュリティ技術の確立
- ビッグデータ解析・エッジコンピューティング：爆発的な量のビッグデータ解析処理を全てクラウドに集約するのではなく、末端に近いセンサ・ゲートウェイに可能な限りオフロードする

→すべてのネットワーク階層の全最適化が必要

本日の小テスト

（配布用紙に学籍番号・氏名を記入し、講義終了後に回収します）

下記設問2問以上の回答と講義の感想（自由記述）

1. 組込みシステムの構成要素、機能・用途、要件を簡単に述べよ。
2. 2013年当時で台数ベースシェアが多い順に並べよ。
   - A) 4/8ビットマイコン
   - B) 16ビットマイコン
   - C) 32ビットマイコン
3. Intel 8051マイコンのデータ幅、アドレス幅、ACC/SP/PCの各レジスタ幅を答えよ。また、ADDのような演算命令の第2オペランドの4つのアドレス形式と、それぞれの命令長（バイト数）を答えよ。
4. IoT時代において従来型の組込みシステムが抱える課題を述べよ。