s4c – s5c

マイコンの最初の最初のハンズオン

苫小牧工業高等専門学校

山本　椋太

始める前に

- このセッションは，ハンズオン形式です．
- 開発環境のインストールを，完了している前提で進めます．
  - まだの方は，至急インストールをお願いします．
  - Slackにインストール方法がございますが，インストーラのファイルサイズが1GB以上です．
  - 会場のWi-Fiだと，ダウンロードが間に合わない可能性が高いです．
- Slackにアップロードした，サンプルコードをダウンロードしておいてください．
  - こちらは，小さなファイルサイズです．

コンピュータの仕組み

- 基本的には，ノイマン型コンピュータと呼ばれる構成をしている.
  - メモリにプログラムを格納してあり，読み出してCPUが実行する構成

- アドレス：
  - 1バイトのデータに1つのアドレスが割り当てられている.
  - コンピュータによって表現できるアドレスのビット数が違う.

MMIO（Memory Mapped I/O）

- 普通，アドレスはメモリと1対1で対応する.

- 特定のアドレスにアクセスすると，
  マイコンの機能レジスタを利用できる.
  - アドレスが，ROMやRAMなどの記憶装置以外の
    意味を持っている.
  - つまり，記憶装置へのアクセスだけでなく
    入出力装置へのアクセスも，アドレスで行う.

メモリマップ（memory map）

- 特にマイコンでは，アドレスで
  メモリの種類や役割が割り当てられている.
- 以下の表はとあるマイコンのメモリマップ

メモリ領域	アドレスの範囲	サイズ	役割
Frash ROM	0x0000 0000 ～ 0x0007 FFFF	512KB	ユーザプログラム・ 定数を格納する
SRAM（内部RAM）	0x2000 0000 ～ 0x2000 FFFF	64KB	変数・スタック領域
ペリフェラル領域	0x4000 0000 ～ 0x4FFF FFFF	-	マイコンの機能を使う場合に 用いる（GPIOなど）
外部メモリ領域	0x6000 0000 ～ 0x9FFF FFFF	-	8MBのフラッシュメモリを使用す るときなど.
システム制御領域	0xE000 0000 ～ 0xE00F FFFF	-	制御レジスタがある.

聞くばかりだとしんどい

- では，プログラムを書いてみたいと思います.

- STM32CubeIDE を起動して，
  プロジェクトを作成します.

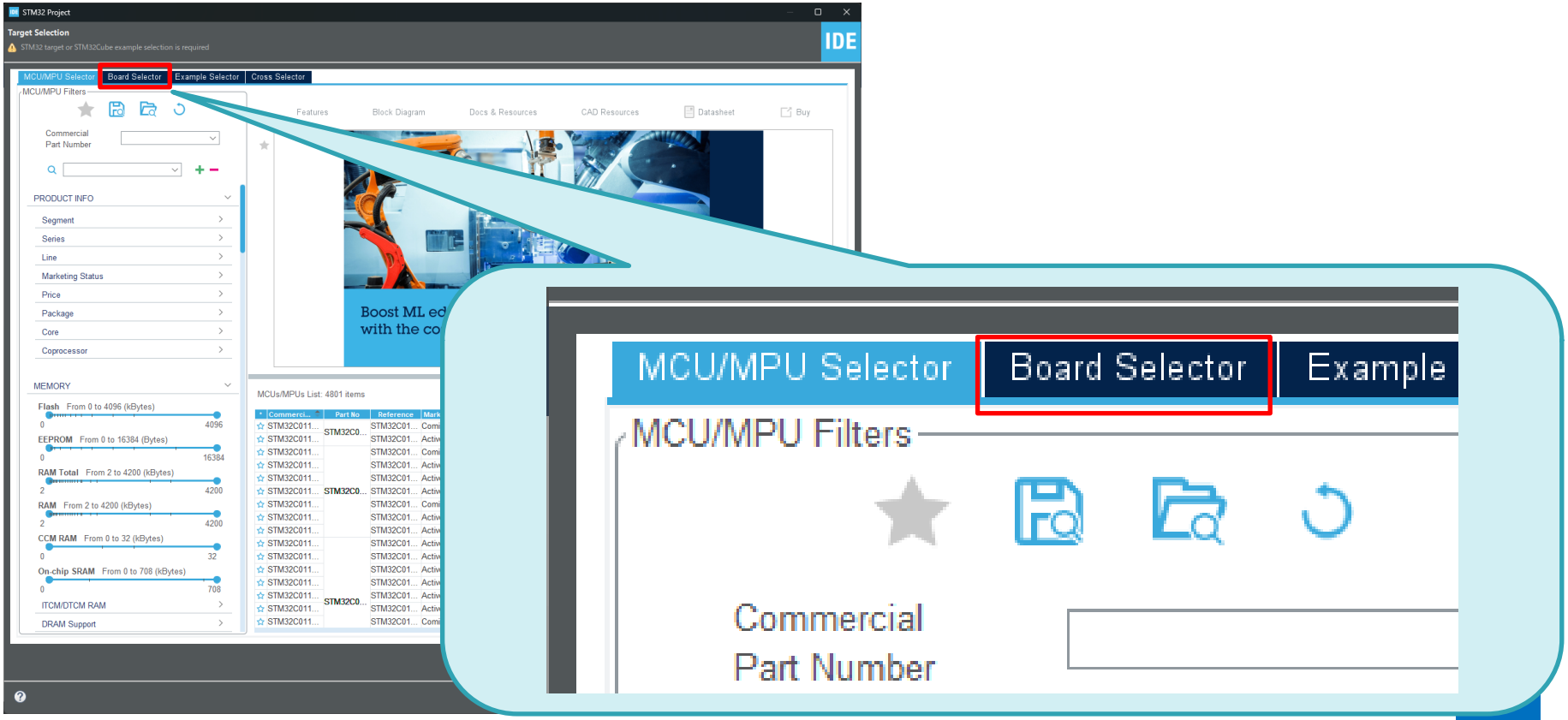
- 以降，インストール手順にも掲載がある
  流れで進めます.

プロジェクトの作成

- 画面左上のアイコン，
  Start New STM32 Project の
  アイコンをクリックする．
  – メニューから，
    Fileを選択すると，
    その中にもある．

プロジェクトの作成

- 以下の画面が立ち上がる.
- 上のタブから，Board Selector をクリック



ボードの選択（1/2）

- 画面左上のCommercial Part Number に NUCLEO-F030R8 と入力する.
- ボードの画像が表示される.

ボードの選択（2/2）

- 表示されたボードの画像をクリックし，ウィンドウ下部のNextボタンをクリック．

プロジェクト名の指定

- Project Name に任意の名前（半角英数推奨）を入力する.
  - 右の画像では test にした.
  - その他はデフォルトの設定で, Finishボタンをクリックする.

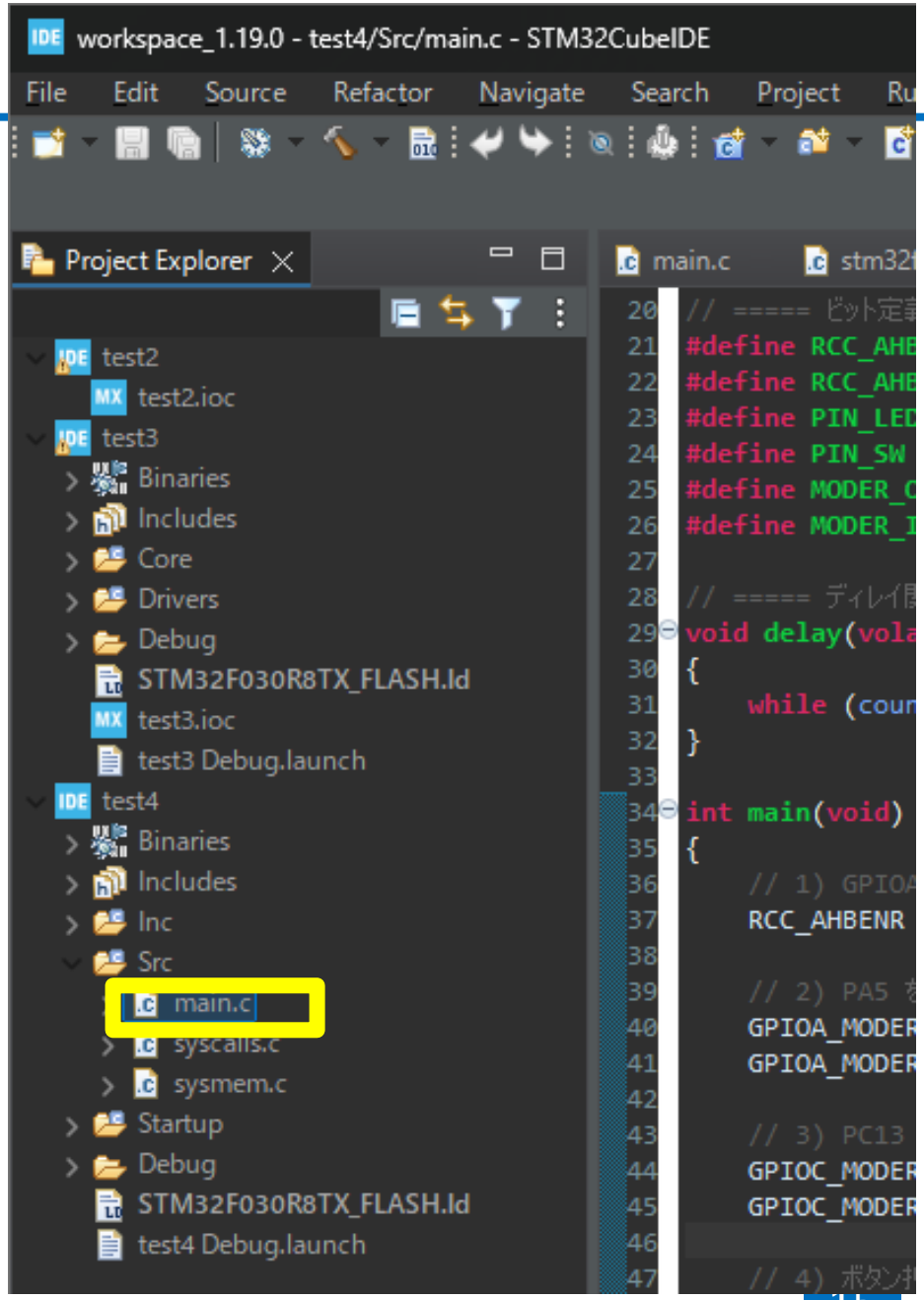
注意

- その後，メッセージボックスが
  表示された場合は，すべてYesを選択する.

プログラミング

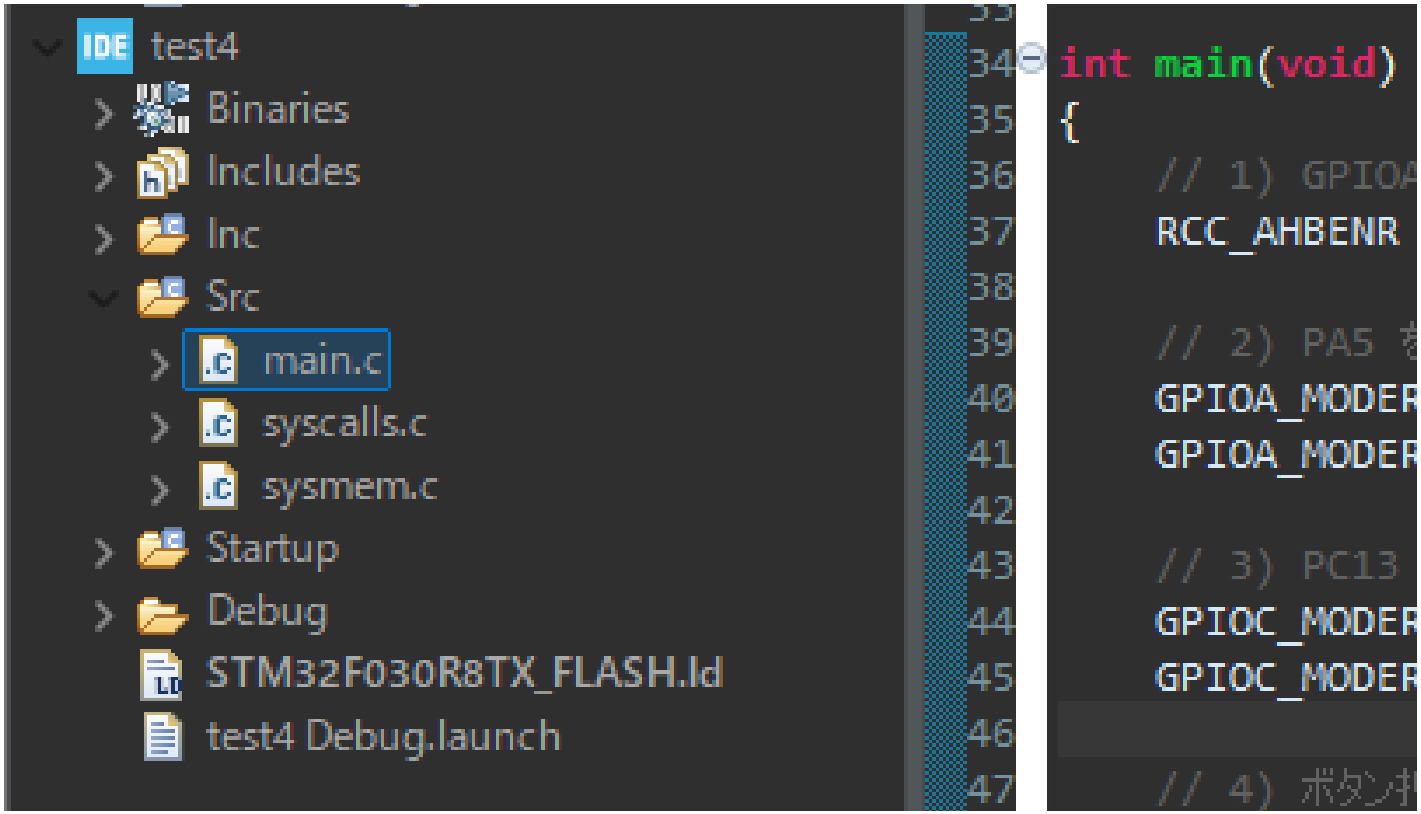
- main.cを
  ダブルクリックで
  開いてください.

- 開いたら,
  すでに書いてある
  プログラムを
  すべて消して
  ください.



注意

- フォルダが図のように生成されない場合，MySTアカウントの紐づけができていないと思われます．



プログラミング

- 配布プログラムから，p1-1.cを
  テキストエディタで開いてください.
  – VSCodeやメモ帳などで開いてください.

- 開いたら，コードをすべてコピーして，
  main.cに貼り付けてください.

- ビルドします.
  – メニューバーの「Project」>「Build All」
  – Ctrl+BでもOK. Macの人は知りません.

プログラミング

- 実行します.
  – メニューバーの「Run」>「Run」

- ボードのLEDが，チカチカするはず.

p1の解説：前提知識（GPIO）

- GPIO（General Purpose Input / Output）を用いる.

- 「高い電圧」または「低い電圧」を出力するための機能である.
  - 高い電圧：3.3Vまたは5Vのことが多い.
  - 低い電圧：0Vのことが多い.

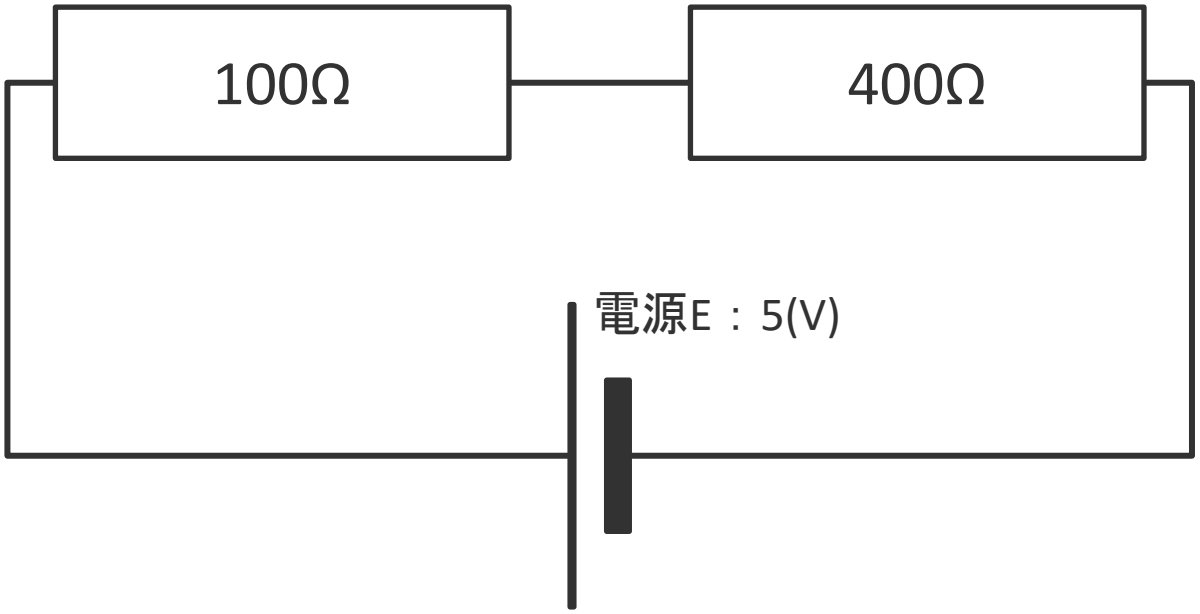
- LEDやスライドスイッチの制御など,「ONかOFF」を取る装置に用いられる.

p1-1の解説：前提知識（電圧と電流）

- 電位：電気的な位置エネルギーの高さ.
  - 通常，基準電位からの高さを指す.
  - 単位はボルト (V)
- 電位差：電気的な位置エネルギーの差
  - どこかとどこか，2点間の差のこと.
  - 電圧と呼ぶと，通常は電位差のこと.
  - 単位はボルト (V)
- 電流：電位が高いところから
  電位が低いところに流れる.
  - 単位はアンペア (A)

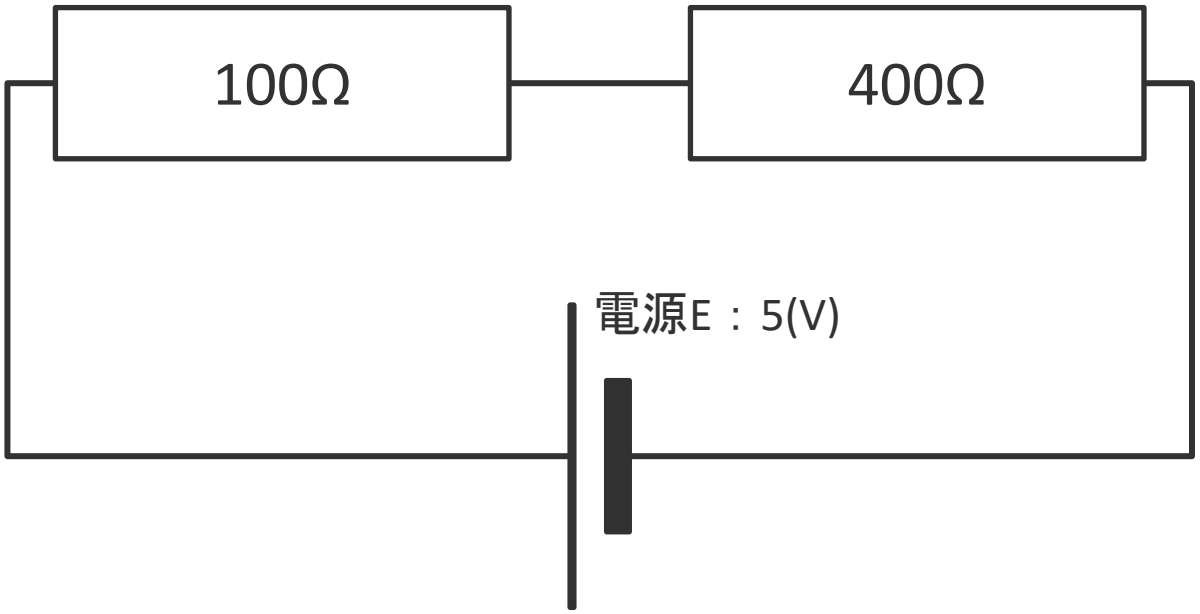
p1-1の解説：前提知識（電圧と電流）

- クイズをします.
  - オームの法則，覚えていますか？
    - V = I × R （電圧 = 電流 × 抵抗）



p1-1の解説：前提知識（電圧と電流）

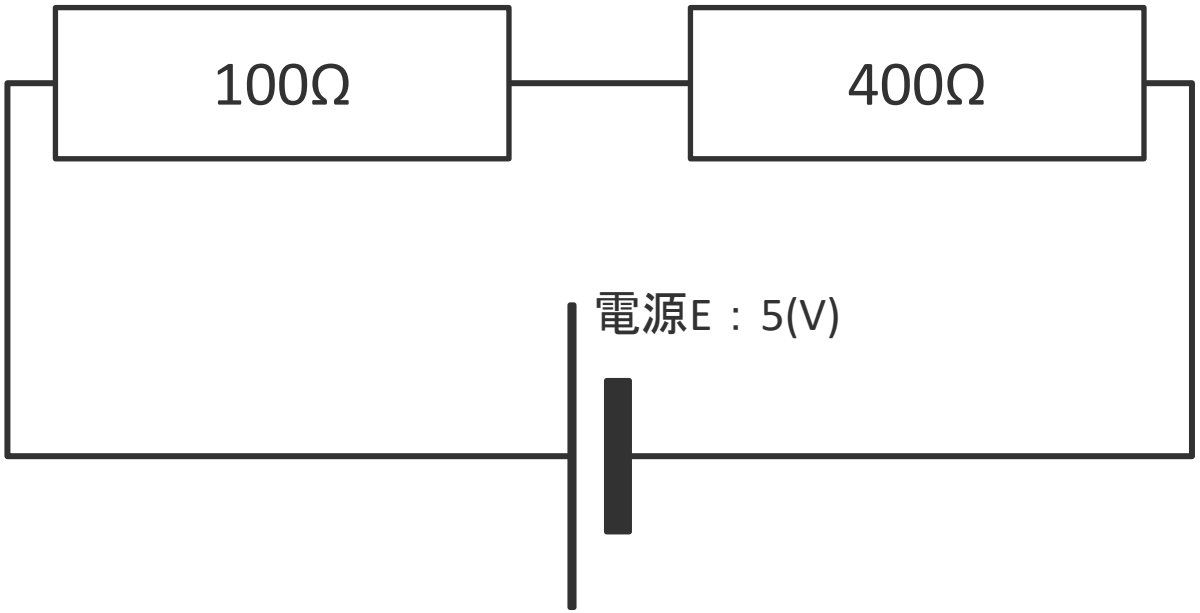
- Q1）この回路全体を流れる電流は何A？
  - V = I × R （電圧 = 電流 × 抵抗）



p1-1の解説：前提知識（電圧と電流）

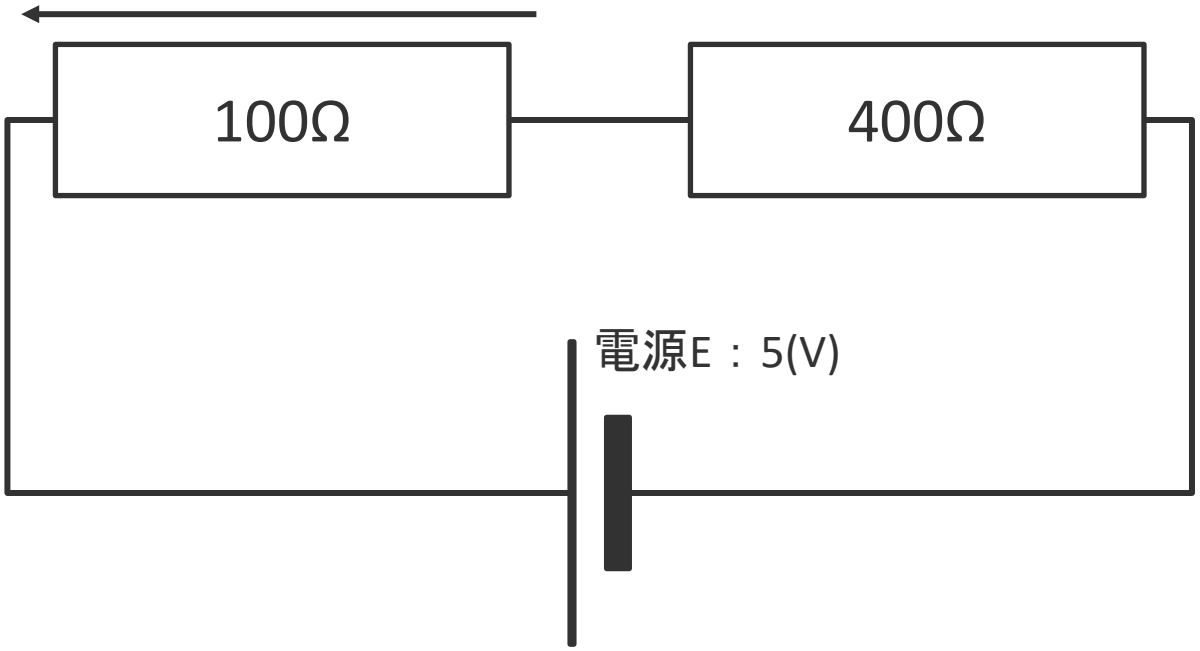
- Q1）この回路全体を流れる電流は何A？
  - V = I × R （電圧 = 電流 × 抵抗）

5 / (100 + 400) = 0.01 A



p1-1の解説：前提知識（電圧と電流）

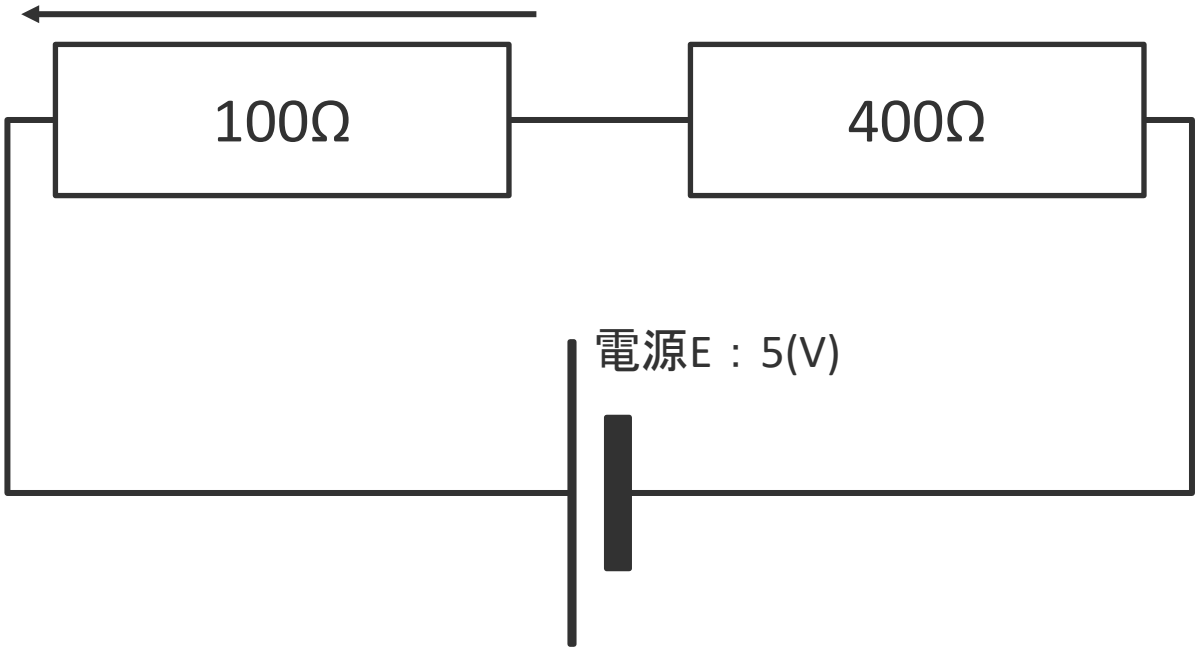
- Q2）100Ωの抵抗の両端の電圧は何V？
  - 矢印の根本を基準にしたときの，矢印の先端の電位の高さはいくつ？
    - V = I × R （電圧 = 電流 × 抵抗）



p1-1の解説：前提知識（電圧と電流）

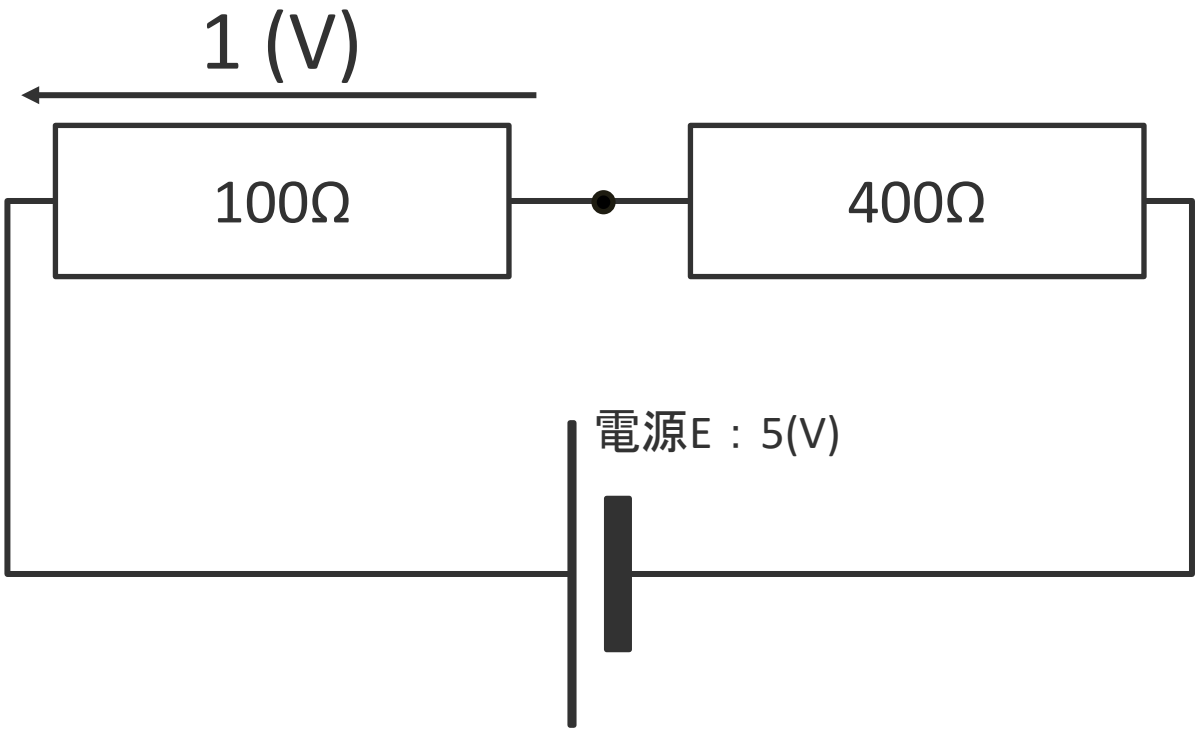
- Q2）100Ωの抵抗の両端の電圧は何V？
  - 矢印の根本を基準にしたときの，矢印の先端の電位の高さはいくつ？
    - V = I × R （電圧 = 電流 × 抵抗）

0.01 × 100 = 1 V



p1-1の解説：前提知識（電圧と電流）

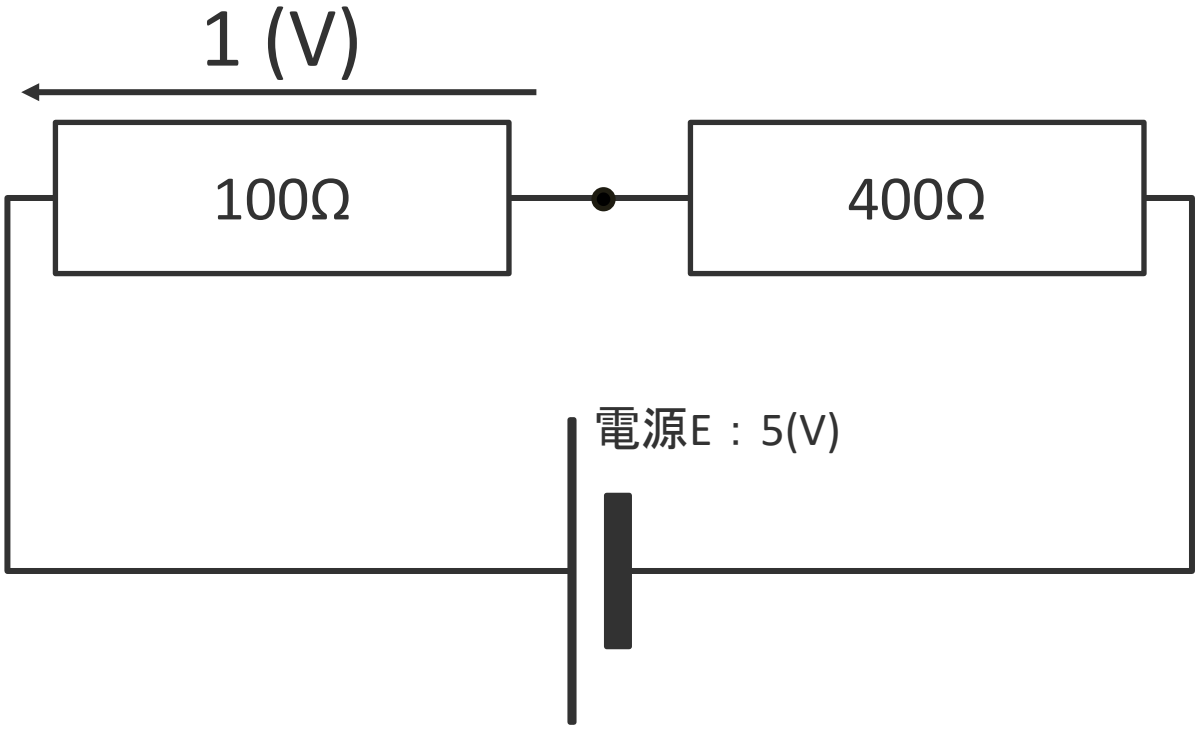
- Q3）100Ωの抵抗と400Ωの抵抗の間の電位は何V？



p1-1の解説：前提知識（電圧と電流）

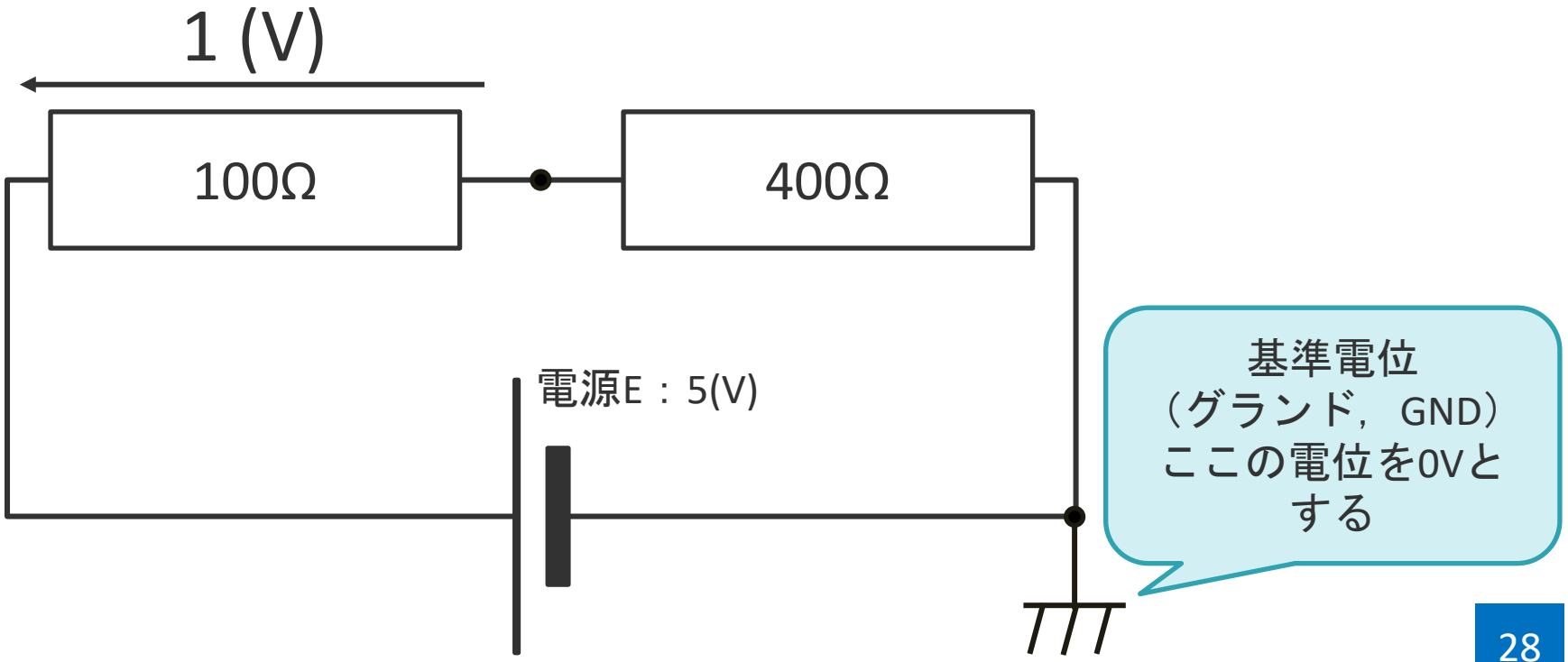
- Q3）100Ωの抵抗と400Ωの抵抗の間の電位は何V？

基準がどこかわからないから，わからない！！
（意地悪してごめんなさい）



p1-1の解説：前提知識（電圧と電流）

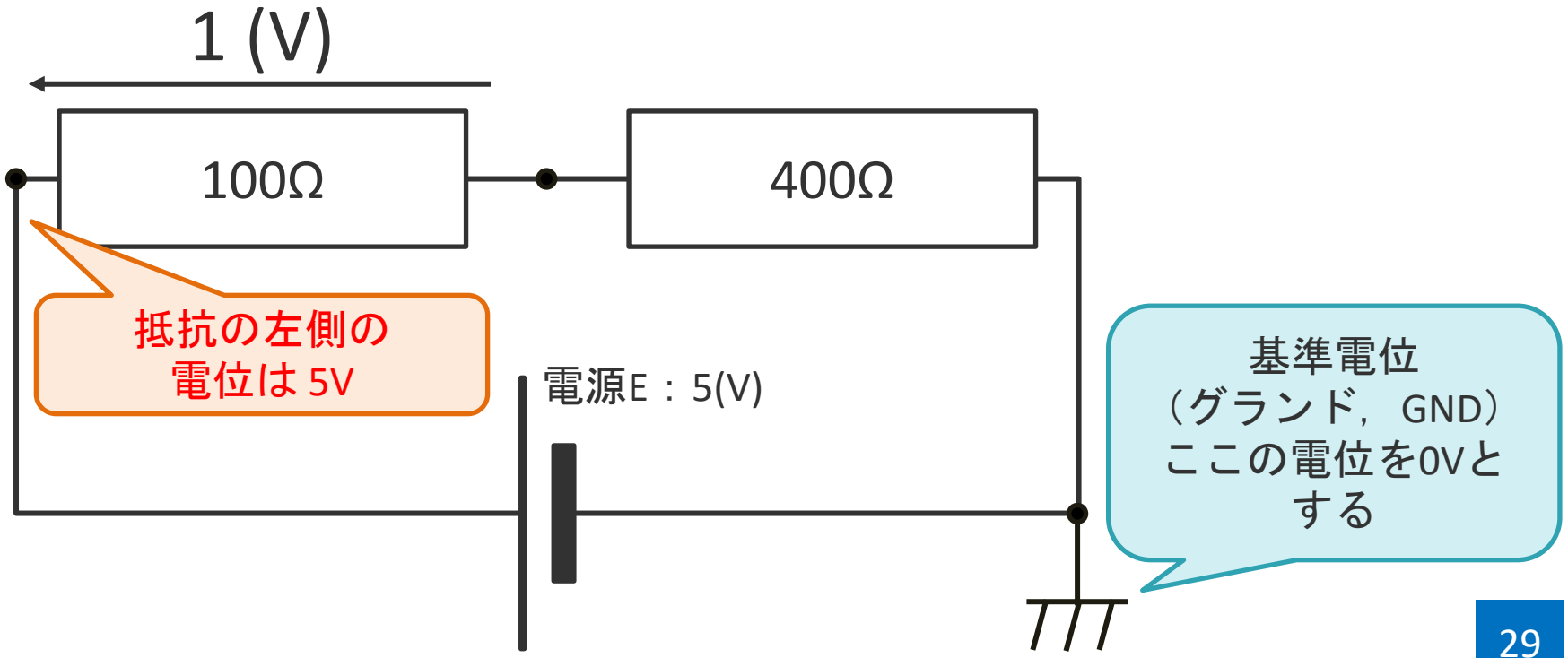
- Q3’）100Ωの抵抗と400Ωの抵抗の間の電位は何V？
  - 基準電位を，電源のマイナス極側とする.



p1-1の解説：前提知識（電圧と電流）

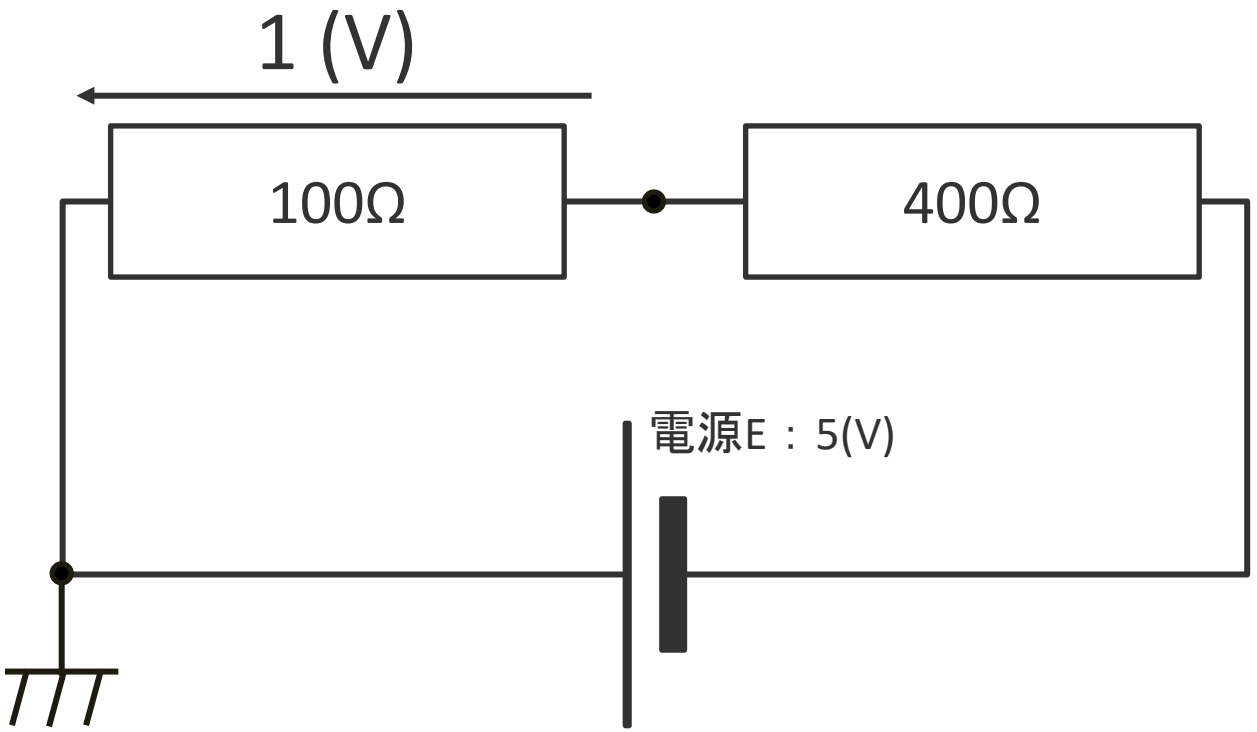
- Q3’）100Ωの抵抗と400Ωの抵抗の間の電位は何V？
  - 基準電位を，電源のマイナス極側とする．

5 – 1 = 4V



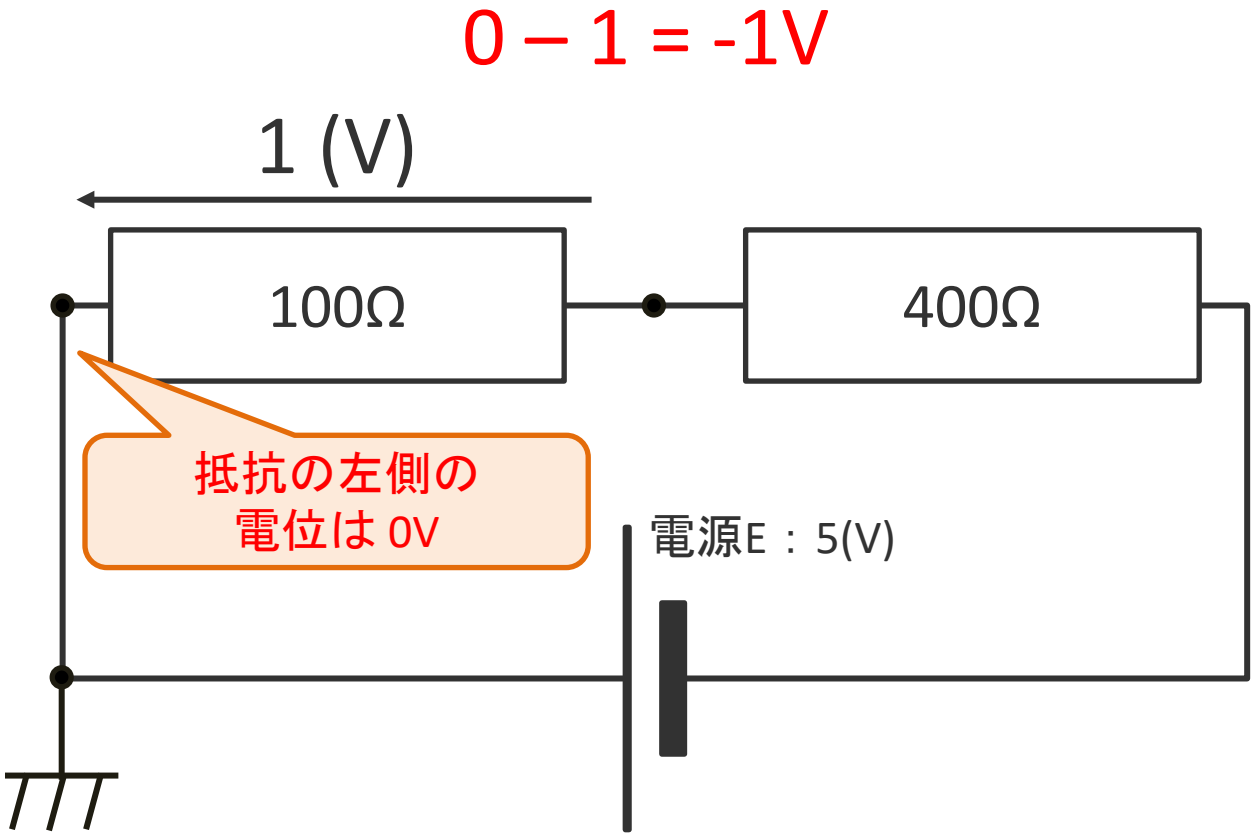
p1-1の解説：前提知識（電圧と電流）

- Q4）100Ωの抵抗と400Ωの抵抗の間の電位は何V？



p1-1の解説：前提知識（電圧と電流）

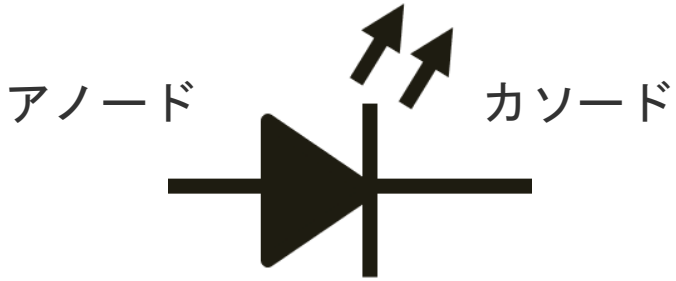
- Q4）100Ωの抵抗と400Ωの抵抗の間の電位は何V？



p1-1の解説：前提知識（LED）

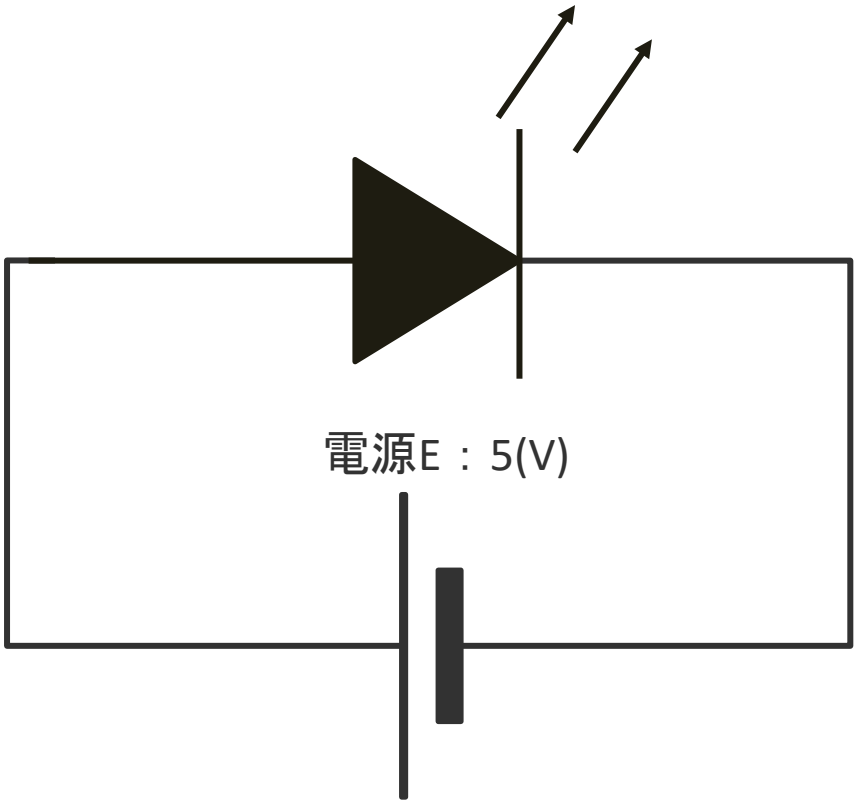
- LED：電流を流すと光る素子のこと.
  - Light Emitting Diode

- 「ダイオード」の1種
  - 一方通行にしか電流を流せない素子
  - 下の図で言えば，左側をアノード，右側をカソードと呼ぶ.
  - アノードからカソードに電流を流すと光る.



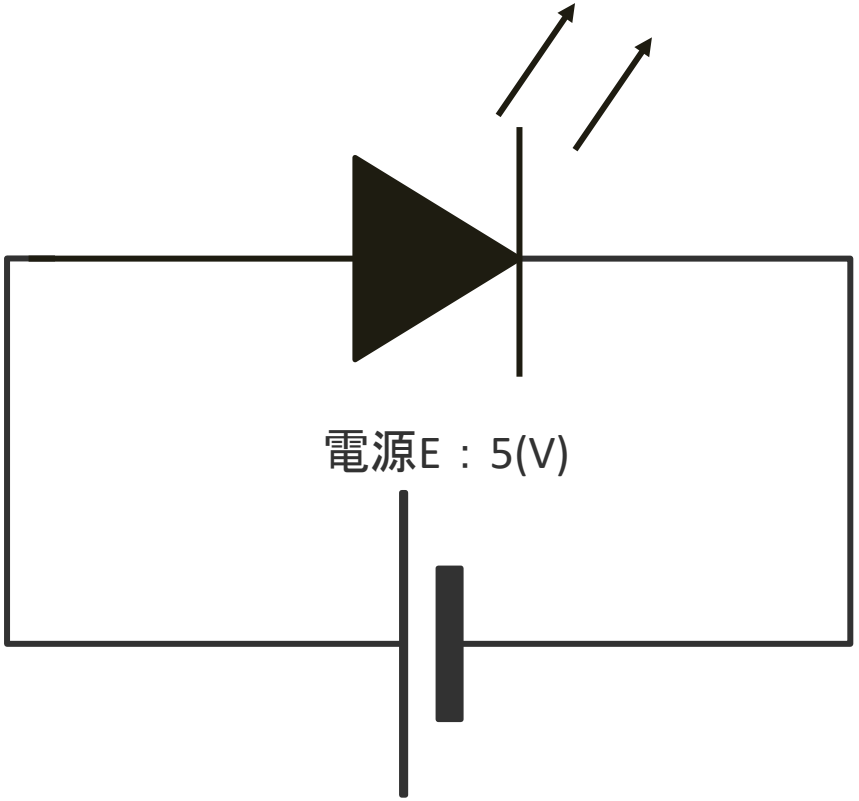
p1-1の解説：前提知識（LED）

- LEDを光らせるには
  - これでアノードからカソードに電流を流せる
  - 完成！！



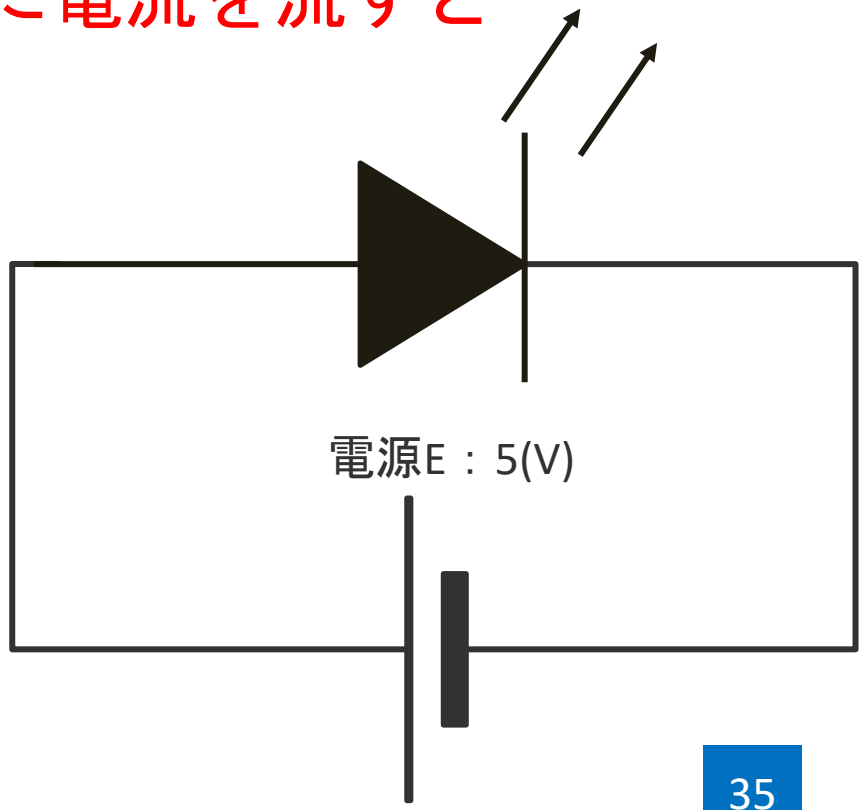
p1-1の解説：前提知識（LED）

- LEDを光らせるには
  - これでアノードからカソードに電流を流せる
  - 完成！！→ダメです



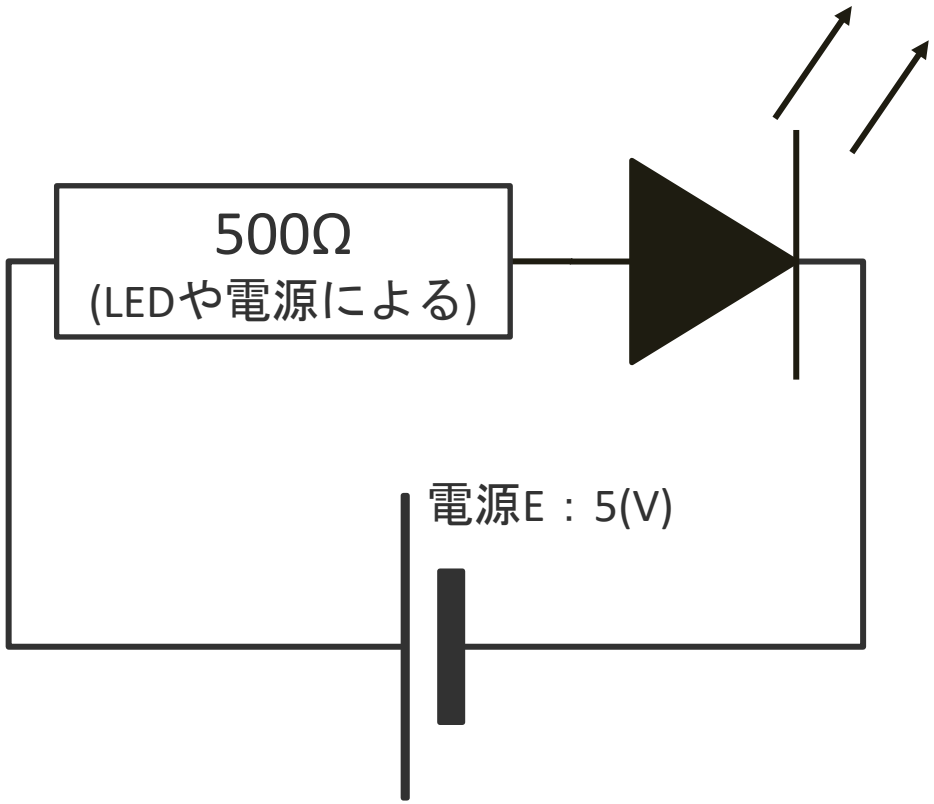
p1-1の解説：前提知識（LED）

- LEDを光らせるには
  - これでアノードからカソードに電流を流せる
  - 完成！！→ダメです
    ダイオードは，順方向に電流を流すと
    理想的には0Ω
    → めっちゃ電流が
    流れてLEDが壊れる…



p1-1の解説：前提知識（LED）

- だから，抵抗を入れて，電流を制限する.



p1の解説：やっと本題

- 7行目から：ベースアドレス定義
  - リファレンスマニュアルでの書かれ方が，ベースアドレスからのオフセットになっている．
  - RCCを設定しないと，GPIOを使えない．
    - 後で簡単に触れます．

```
// ===== ベースアドレス：ここからオフセットを与える形にしていきたい．　=====
#define RCC_BASE 0x40021000UL
#define GPIOA_BASE 0x48000000UL
```

- RM0360：（40/775）

使っている機能（RM0360参照）

- GPIOA
  - General Purpose Input/Output A
  - 汎用入出力レジスタの A
- RCC：詳しい説明は割愛します.
  - Reset and Clock Control
  - リセットの設定やクロックの設定ができる.
  - ある機能に対してクロックを有効化する.
    - つまり，ある機能の使用/不使用を切り替える.
    - この設定がないと，使いたい機能を使えない.

GPIOA

- RM0360 Table 2（p.40）より，
  0x4800 0000 から 0x4000 03FF が
  GPIOA関係のレジスタである.
  - RM0360：（40/775）

- RM0360 8章4節からがレジスタの説明
  - 動作の説明は8.4以前にあります.
  - RM0360：（134/775）

ODRとBSRR

- ODR（RM0360：（136/775））16ビットは,
  - 0: 出力電圧0[V]
  - 1: 出力電圧5[V]
- BSRR（RM0360：（137/775））32ビットは,
  - 上位16ビット
    - 0：変化なし
    - 1：出力電圧0[V]
  - 下位16ビット
    - 0：変化なし
    - 1：出力電圧5[V]

ODRとBSRR

- どちらも出力電圧を変化させるが…

- BSRRは，ハードウェア的に
  特定のビットの値のみを変化させることができる.

- ODRは，
  - 良くも悪くも，全ビットを同時に書き換える.
  - 1ビットだけで出力値を決めるため，
    現在の値からの変化などを書こうと思うと，
    使いやすい.

ODRとBSRR

- 例）Lチカ

```
while (1)
{
	GPIOA_BSRR = (1u << PIN_LED);
	delay(500000);
	GPIOA_BSRR = (1u << (PIN_LED + 16));
	delay(500000);
}
```

```
while (1)
{
	GPIOA_ODR = ~GPIOA_ODR;
	delay(500000);
}
```

ODRとBSRR

- 例）Lチカ

どんな値にするか，
確実に決まっているなら，
こちらのほうが間違いがない．
が，行数はこちらのほうが多い．

```
while (1)
{
      GPIOA_BSRR = (1u << PIN_LED);
      delay(500000);
      GPIOA_BSRR = (1u << (PIN_LED + 16));
      delay(500000);
}
```

行数は少ないが，
PA5以外のビットも反転するため，
想定外の問題が起こるかも．
けど，現在の値を使った処理は書きやすい．

```
while (1)
{
      GPIOA_ODR = ~GPIOA_ODR;
      delay(500000);
}
```

p2.c

- p1-2.cは，スイッチがONになると，
  LEDが光るプログラムです.

- そのままではコンパイルエラーです.
  – main.cに上書きしてください.

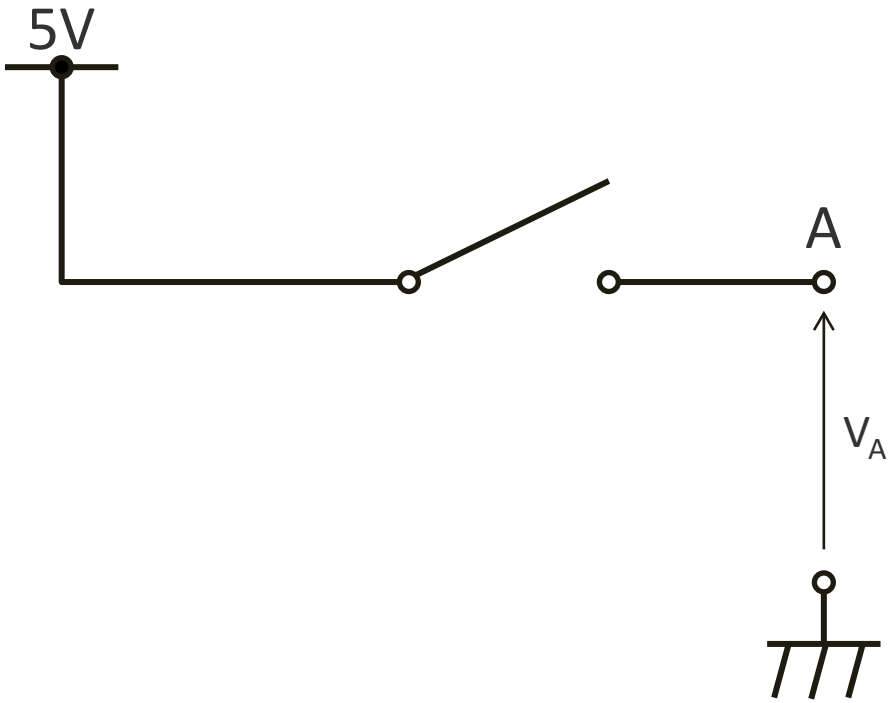
- コメント欄に当てはまる値を，
  考えてみましょう.
  – リファレンスマニュアルから探してみましょう.

スイッチの動作原理

- スイッチはONだと2進数の1,
  OFFだと2進数の0？
  → そうとも限らない.

スイッチと電位

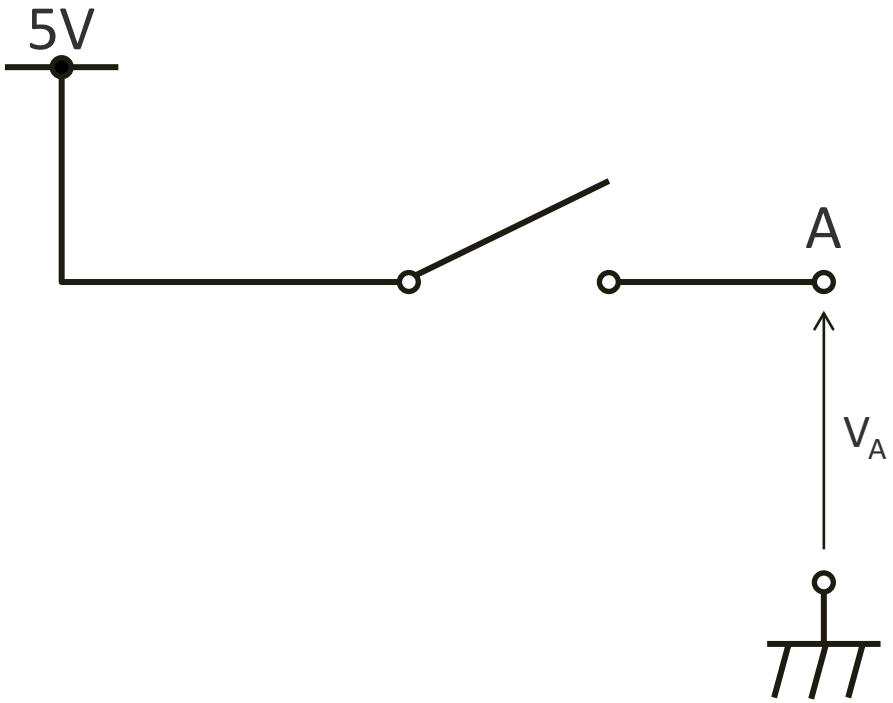
- 質問です．点Aの電位V_A[V]は？



スイッチがONのとき：　[V]
スイッチがOFFのとき：　[V]

スイッチと電位

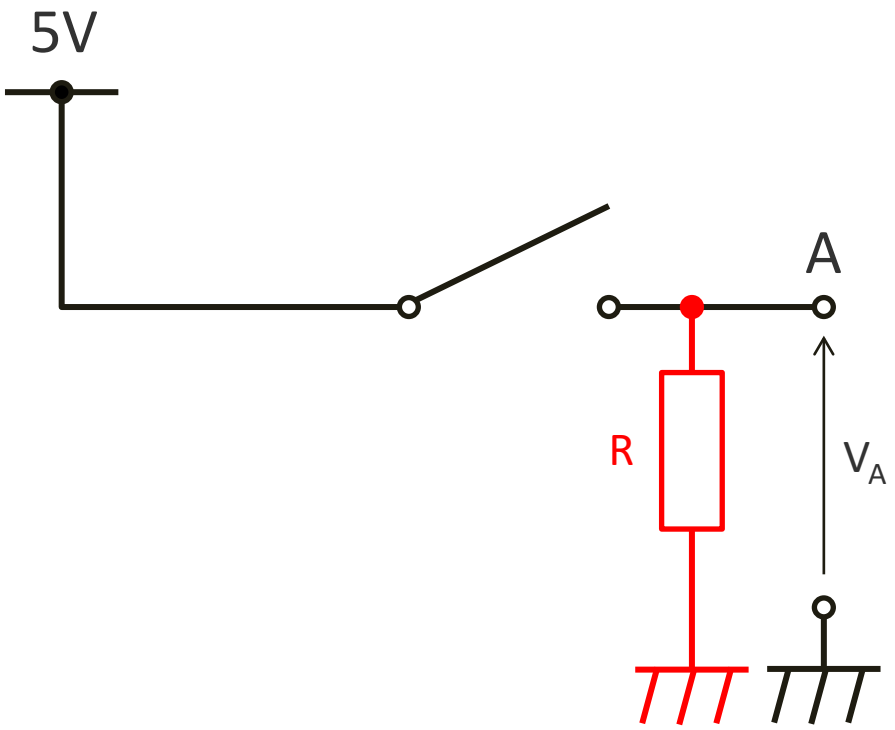
- 質問です．点Aの電位V_A[V]は？



スイッチがONのとき：5 [V]
スイッチがOFFのとき：？[V] → 不定！

スイッチと電位：プルダウン

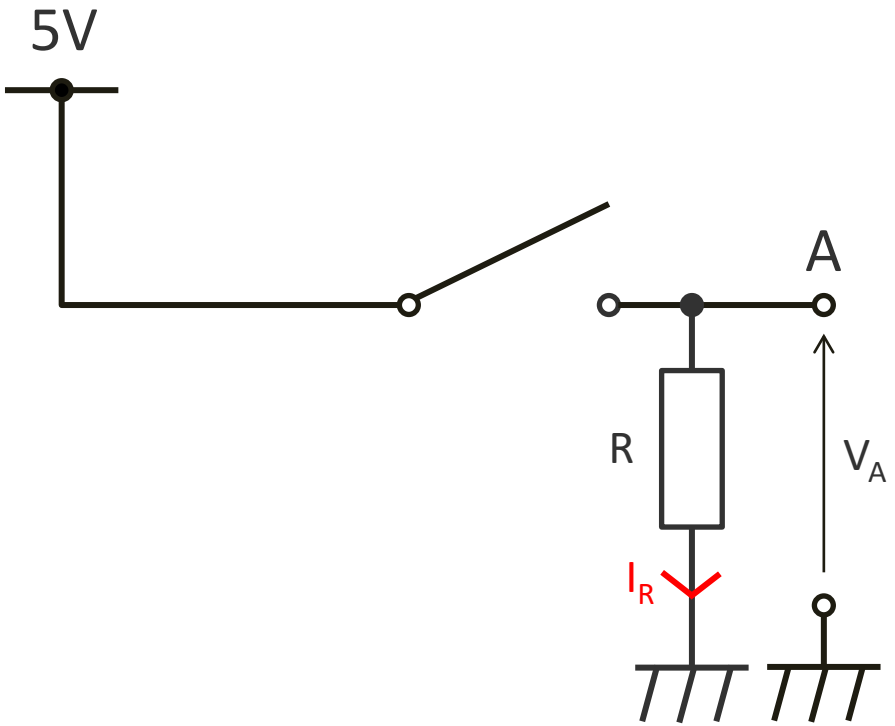
- どうすればいい？



出力側に，グランドと接続している抵抗Rを追加

スイッチと電位：プルダウン

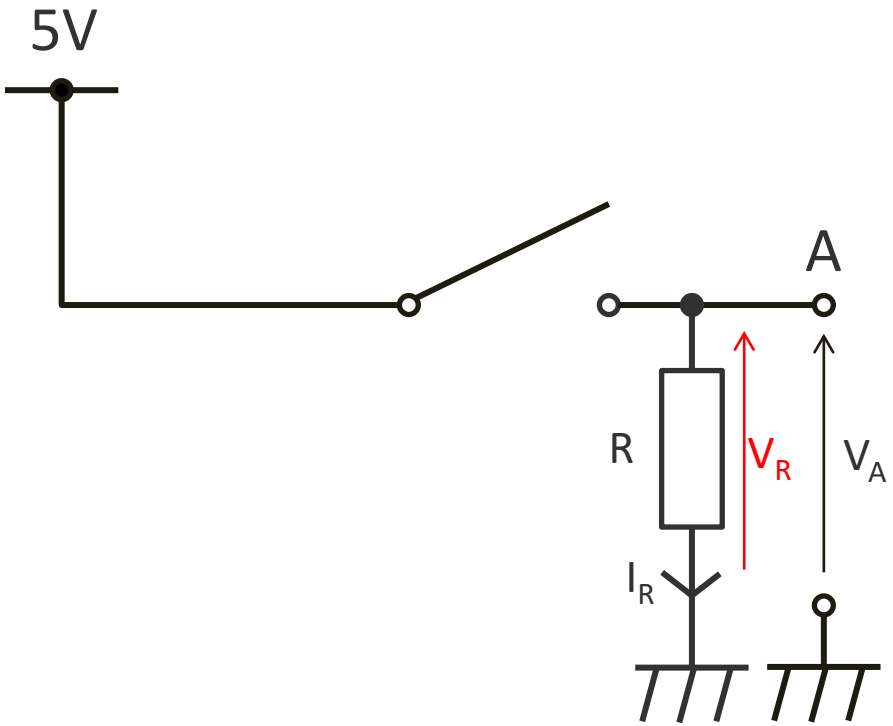
- どうすればいい？



スイッチがOFFのとき，抵抗Rに流れる電流I_Rは0[A].
→ 電位差を生じさせる要素がどこにもないため.

スイッチと電位：プルダウン

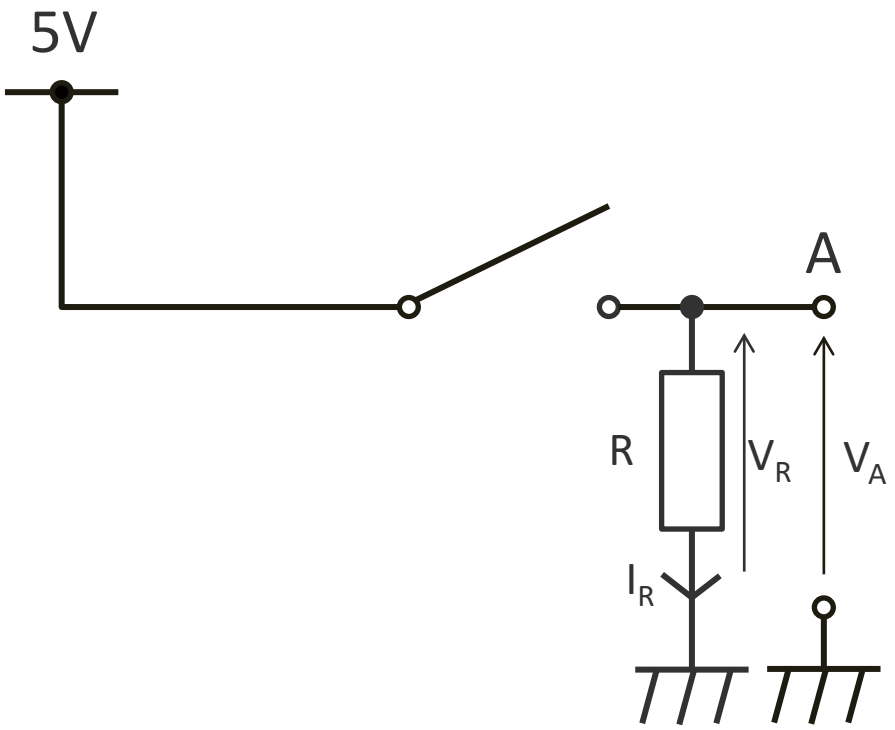
- どうすればいい？



抵抗Rの両端の電圧（電位差）V_Rは 0[V]
（∵オームの法則．V = IR より，I = 0 なら，V = 0である．）

スイッチと電位：プルダウン

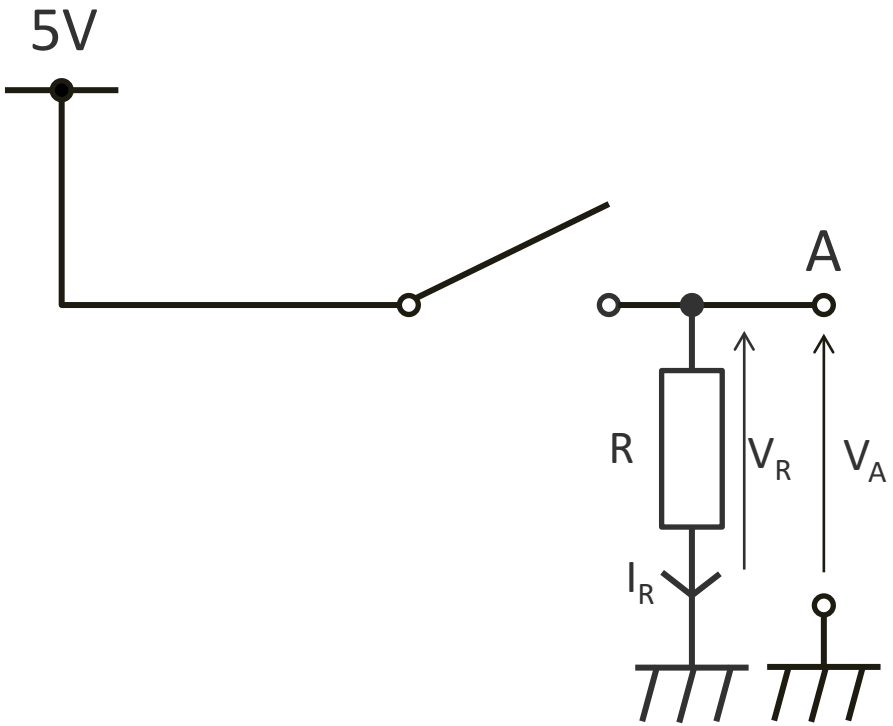
- 不定にしない方法



抵抗Rの両端で電位差が無いということは，
抵抗Rの「上側」と「下側」は同じ電位である．
→ 抵抗の下側は，電位 0[V] だから，上側も電位 0[V]

スイッチと電位：プルダウン

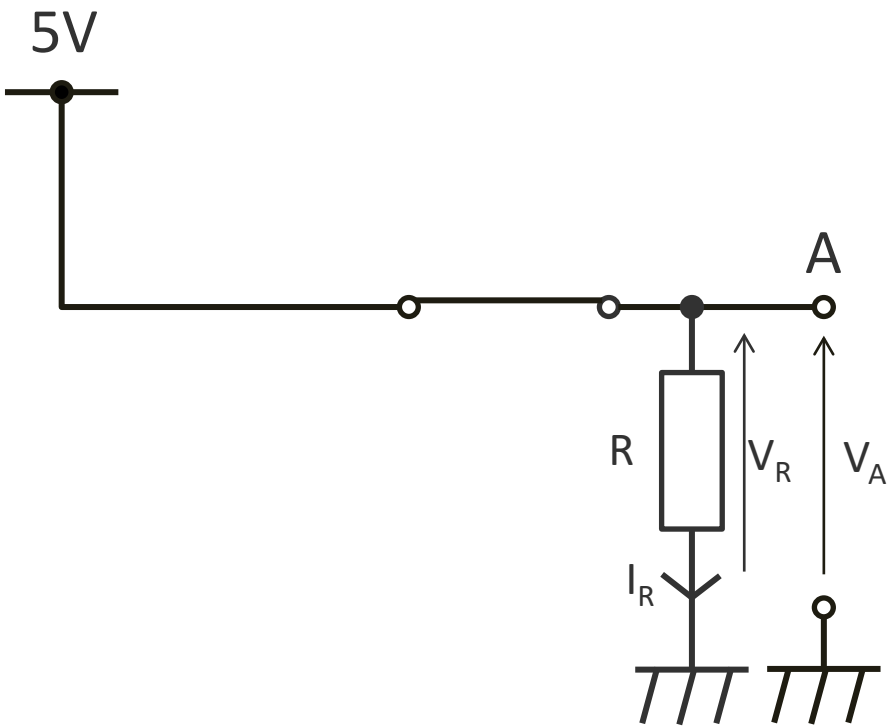
- 不定にしない方法



したがって，スイッチがOFFのとき，
V_Aは，0[V]になる．→ 不定ではなくなる．

スイッチと電位：プルダウン

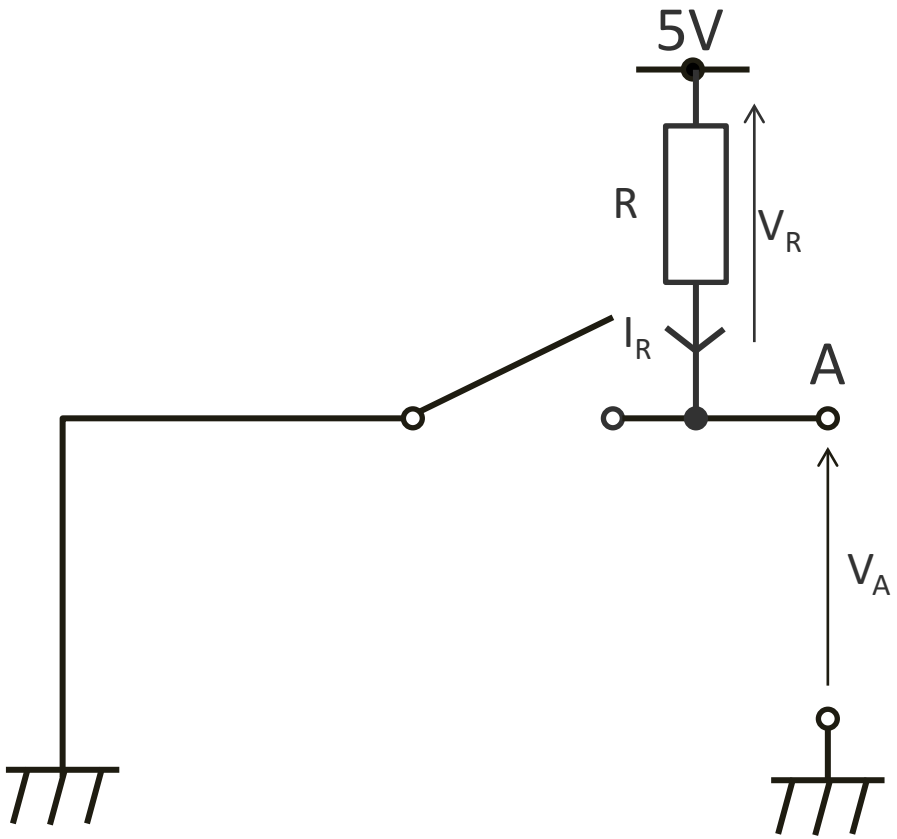
- 不定にしない方法



スイッチがONのときは，V_Aは5Vになる．

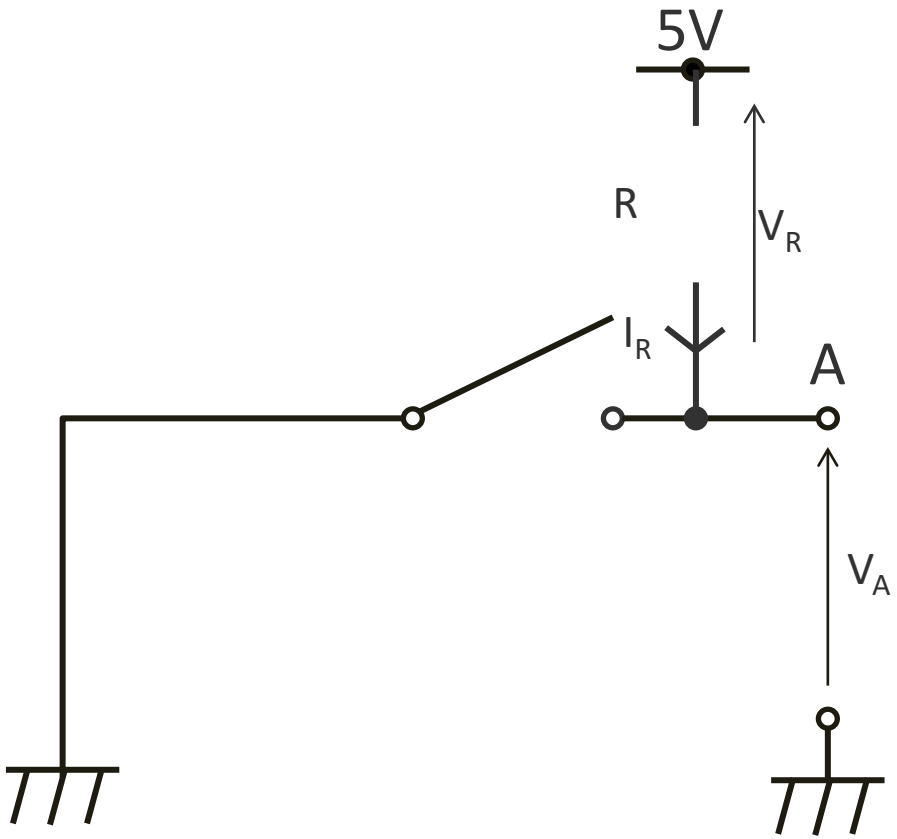
スイッチと電位：プルアップ

- 回路図は以下の通り.
- 理屈は同じだが，負論理（ONで0）である.
- 慣例的にはこちらが使われることが多い.



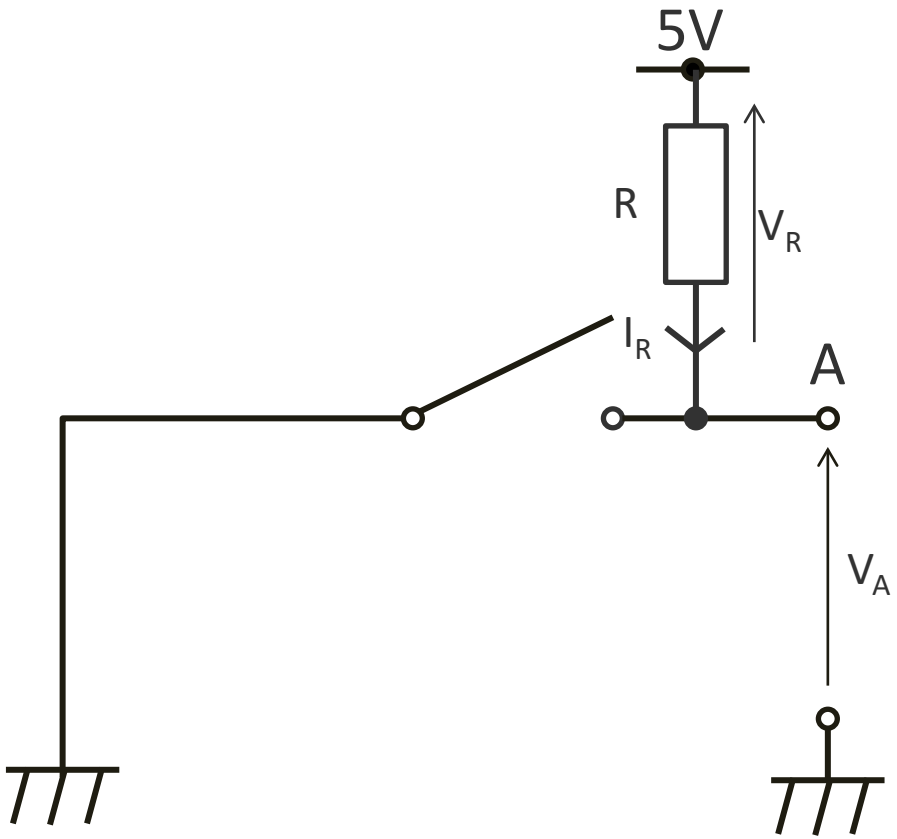
スイッチと電位：プルアップ

- 回路図は以下の通り.
- 理屈は同じだが，負論理（ONで0）である.
- 慣例的にはこちらが使われることが多い.



スイッチと電位：プルアップ

- 回路図は以下の通り.
- 理屈は同じだが，負論理（ONで0）である.
- 慣例的にはこちらが使われることが多い.



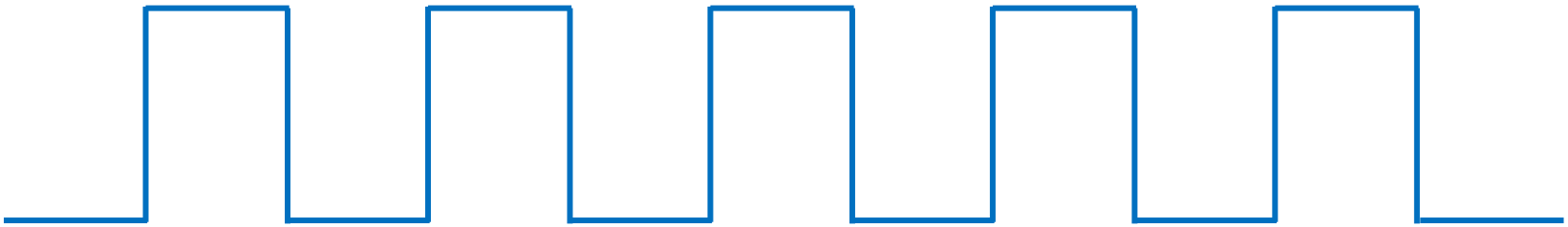
ハードウェアタイマ

- クロック信号による正確なタイマのこと.
  - タイマは非常に重要.
    - リアルタイム性の保証,
    - OSにおける時間管理，周期実行など.

- ソフトウェアタイマ
  - ループなどで，擬似的にタイマを作る.
  - 目測でそこそこまでは作れる？
    - マイクロ秒？人間の限界にチャレンジ！

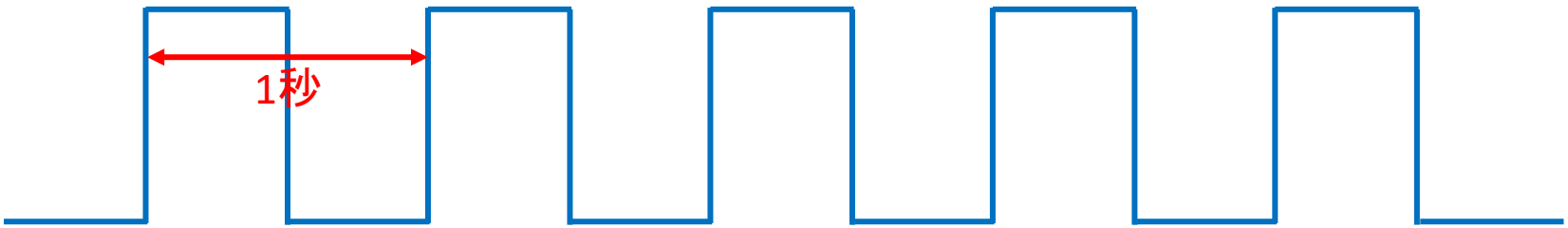
一般的なハードウェアタイマの仕組み

- 質問：1Hzのクロック信号があるとき，
  - 周期は何秒か？
  - では2周期ちょうど経過すると何秒か．
  - つまり，クロックの立ち上がりの数を数える
    → カウンタを利用すると…？



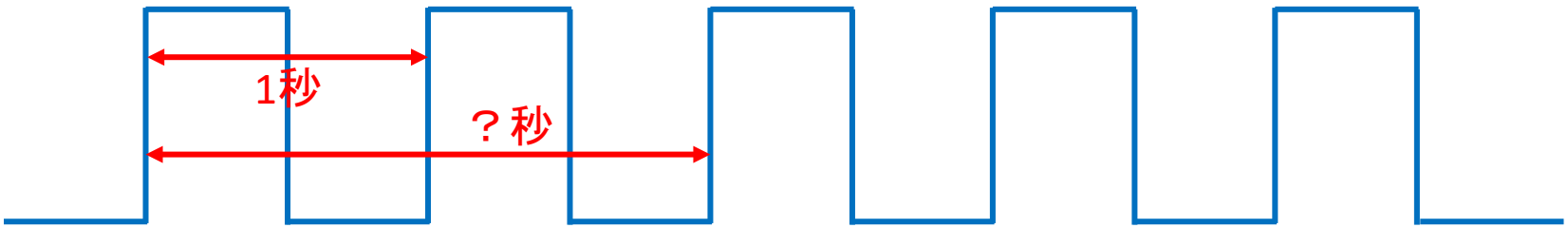
一般的なハードウェアタイマの仕組み

- 質問：1Hzのクロック信号があるとき，
  - 周期は何秒か？
  - では2周期ちょうど経過すると何秒か．
  - つまり，クロックの立ち上がりの数を数える
    → カウンタを利用すると…？



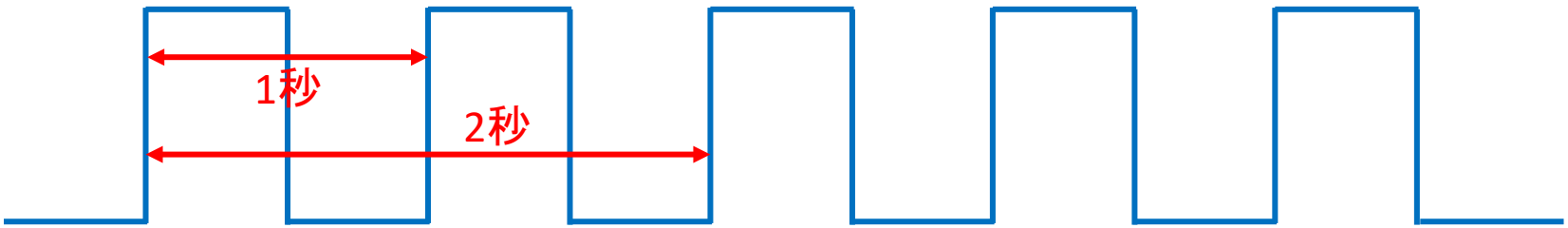
一般的なハードウェアタイマの仕組み

- 質問：1Hzのクロック信号があるとき，
  - 周期は何秒か？
  - では2周期ちょうど経過すると何秒か．
  - つまり，クロックの立ち上がりの数を数える
    → カウンタを利用すると…？



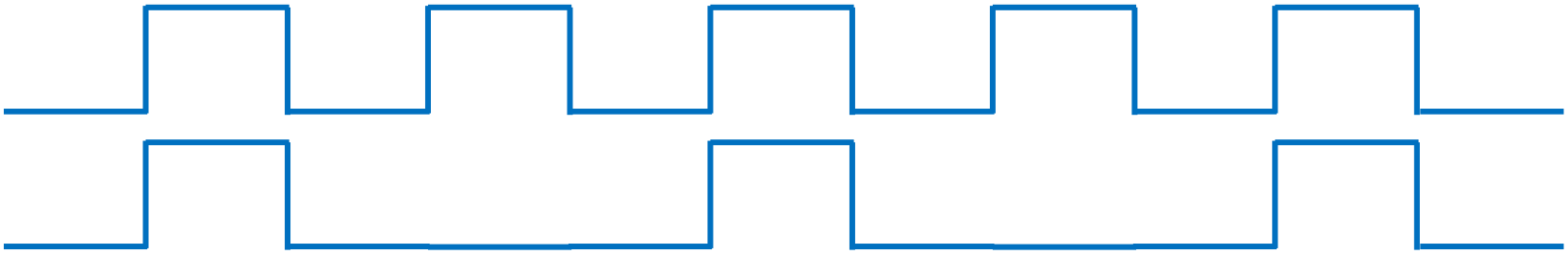
一般的なハードウェアタイマの仕組み

- 質問：1Hzのクロック信号があるとき，
  - 周期は何秒か？
  - では2周期ちょうど経過すると何秒か．
  - つまり，クロックの立ち上がりの数を数える
    → カウンタを利用すると…？



一般的なハードウェアタイマの仕組み

- 分周（プリスケール）
  - 高周波数のクロック信号の周波数を定数で割る
    - 例）12MHzを4で分周すると3MHzになる.
  - 高周波数の場合，カウンタのビット数が不足する可能性がある.
    - つまり，8ビットのカウンタだとすると，0から255までしかカウントできない.
      → 周波数が高いと，カウント値が不足しうる.



一般的なハードウェアタイマの仕組み

- 目標の時間の作り方タイマの仕組み
  - タイマの初期設定をする
    - 分周などの設定
    - コンペアマッチの値の設定
      → カウンタの値がいくつになったら，
      指定時間経過したかを設定する.
  - タイマスタート・ストップ
    - タイマスタートで，カウンタが，
      クロックの立ち上がり回数を数え始める.
    - コンペアマッチの値とカウンタの値が一致したら，
      それを知らせる信号が変化する.
    - タイマの使用が終わったら，タイマストップする.

一般的なハードウェアタイマの仕組み

- コンペアマッチの値の決め方
  1. タイマの入力となるクロック周波数を確認する.
  2. コンペアマッチの値
     （= 何回立ち上がりエッジを数えたら）
     を考える.
     注意：0から数えるため,
     最後に1引くことを忘れない.
  3. もし，カウンタのビット数で収まらない
     コンペアマッチ値の場合は，分周する.
     （分周なしでも，分周率の設定は必要）

p3.c

TIM14（398/775）

- TIMx_CNT
  - Counter Register
- TIMx_PSC
  - Prescaler
- TIMx_ARR
  - Auto Reloaded Register