gcc（ジーシーシー）をいれておこう！

Finder
→ アプリケーション
→ ユーティリティー
→ ターミナル
→ gcc （エンター）
* no input files とエラーがでてもOK!

15コマ集中講義 ITブートキャンプ Part6

マシン語とOS

神山まるごと高専 技術教育統括ディレクター

福野泰介 @taisukef

一日一創

ITブートキャンプ カリキュラム

4/1	4/2	4/3	4/4	4/5	4/8
-	-	-	-	演習時間 Q&Aコーナー	開発時間
-	-	サイバーセキュリティ IchigoJamでネットワークとプロトコル	-	まるごとアイデアソン	開発時間
電子工作 IchigoJamはんだづけ	計測と制御 IchigoJam サーボ&センサー	マシン語とOS IchigoJam Armマシン語	ウェブアプリ開発 HTML+CSS+JavaScript	まるごとハッカソン	まるごとプレゼン *
プログラミング IchigoJamプログラミング	演習時間 * IchigoJamで自由工作	C言語 gccとIchigoJamをいじる	AIとVR * 自由に作ってみよう		

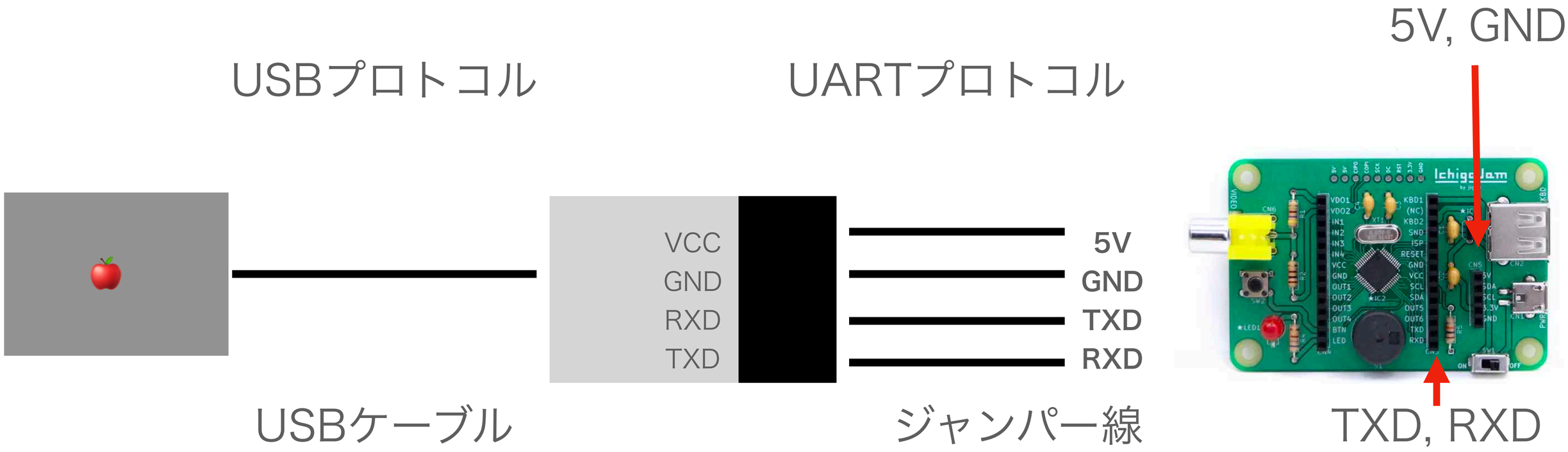
1限: 9:00-10:30, 2限: 10:45-12:15, 3限: 13:15-14:45, 4限: 15:00-16:30

* レポート計3回

IcigoJamは1秒に
何回計算できる？

実験！

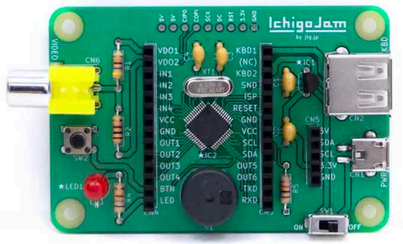
MacとIchigoJamをつなぐ



https://ichigojam.github.io/IchigoTerm/

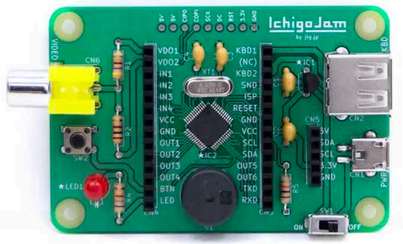
```
10  A=0:CLT
20  A=A+1
30  IF  A<10000  GOTO20
40  ?TICK()/6
```

IchigoJam R



```
10  A=0:CLT
20  A=A+1+0
30  IF  A<10000  GOTO20
40  ?TICK()/6
```

IchigoJam R

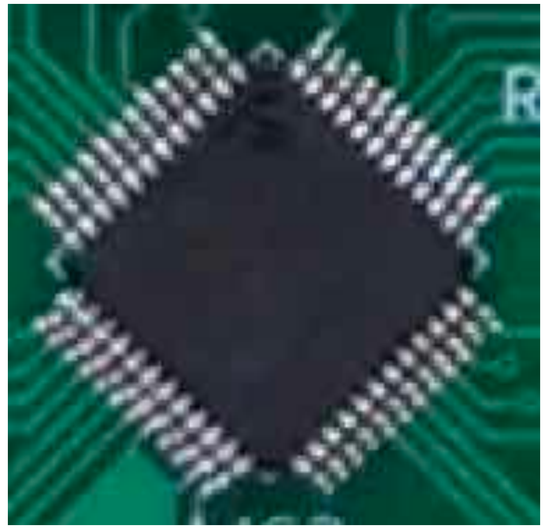


2.2秒で1万回

・・・約4,500回/秒

ん、遅くない！？

コンピューターの仕組み



ハードウェア

300円のCPU

GD32VF103

by GigaDevice

↑

ソフトウェアを書き込んで動かす

作り方は、CPUの**データシート**に書いてある

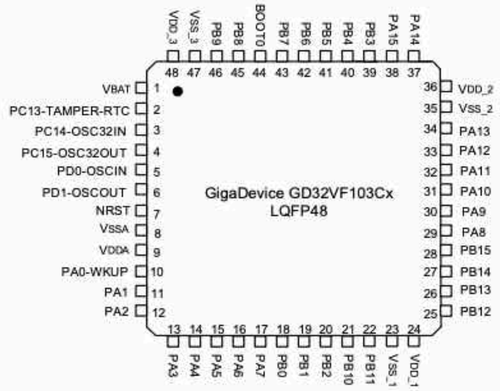
GigaDevice Semiconductor Inc.

GD32VF103
RISC-V 32-bit MCU

Datasheet

GigaDevice GD32VF103 Datasheet

Figure 2-4. GD32VF103Cx LQFP48 pinouts



GD32VF103 Datasheet

2.4. Memory map

Table 2-3. GD32VF103 memory map

Pre-defined Regions	Bus	Address	Peripherals
External device	AHB	0xA000 0000 - 0xA000 0FFF	EXMC_SWREG
External RAM	AHB	0x9000 0000 - 0x9FFF FFFF	Reserved
		0x7000 0000 - 0x8FFF FFFF	Reserved
		0x6000 0000 - 0x6FFF FFFF	EXMC - NOR/PSRAM/SRAM
Peripheral	AHB	0x5000 0000 - 0x5003 FFFF	USBFS
		0x4008 0000 - 0x4FFF FFFF	Reserved
		0x4004 0000 - 0x4007 FFFF	Reserved
		0x4002 BC00 - 0x4003 FFFF	Reserved
		0x4002 B000 - 0x4002 BBFF	Reserved
		0x4002 A000 - 0x4002 AFFF	Reserved
		0x4002 8000 - 0x4002 9FFF	Reserved
		0x4002 6800 - 0x4002 7FFF	Reserved
		0x4002 6400 - 0x4002 67FF	Reserved
		0x4002 6000 - 0x4002 63FF	Reserved
		0x4002 5000 - 0x4002 5FFF	Reserved
		0x4002 4000 - 0x4002 4FFF	Reserved
		0x4002 3C00 - 0x4002 3FFF	Reserved
		0x4002 3800 - 0x4002 3BFF	Reserved
		0x4002 3400 - 0x4002 37FF	Reserved
		0x4002 3000 - 0x4002 33FF	CRC
		0x4002 2C00 - 0x4002 2FFF	Reserved
		0x4002 2800 - 0x4002 2BFF	Reserved
		0x4002 2400 - 0x4002 27FF	Reserved
		0x4002 2000 - 0x4002 23FF	FMC
		0x4002 1C00 - 0x4002 1FFF	Reserved
		0x4002 1800 - 0x4002 1BFF	Reserved
		0x4002 1400 - 0x4002 17FF	Reserved
		0x4002 1000 - 0x4002 13FF	RCU
		0x4002 0C00 - 0x4002 0FFF	Reserved
		0x4002 0800 - 0x4002 0BFF	Reserved
		0x4002 0400 - 0x4002 07FF	DMA1
		0x4002 0000 - 0x4002 03FF	DMA0
		0x4001 8400 - 0x4001 FFFF	Reserved
		0x4001 8000 - 0x4001 83FF	Reserved

14

3.4. Boot modes

At startup, boot pins are used to select one of three boot options:

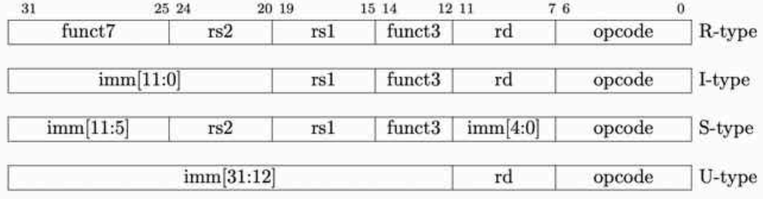
- Boot from main flash memory (default)
- Boot from system memory
- Boot from on-chip SRAM

The boot loader is located in the internal boot ROM memory (system memory). It is used to reprogram the Flash memory by using USART0 (PA9 and PA10), USART1 (PD5 and PD6), USBFS in device mode (PA9, PA11 and PA12). It also can be used to transfer and update the Flash memory code, the data and the vector table sections.

2.2 Base Instruction Formats

In the base ISA, there are four core instruction formats (R/I/S/U), as shown in Figure 2.2. All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction address misaligned exception is generated on a taken branch or unconditional jump if the target address is not four-byte aligned. No instruction fetch misaligned exception is generated for a conditional branch that is not taken.

The alignment constraint for base ISA instructions is relaxed to a two-byte boundary when instruction extensions with 16-bit lengths or other odd multiples of 16-bit lengths are added.



file:///Users/fukuno/Downloads/GD32VF103_Datasheet_Rev1.0.pdf

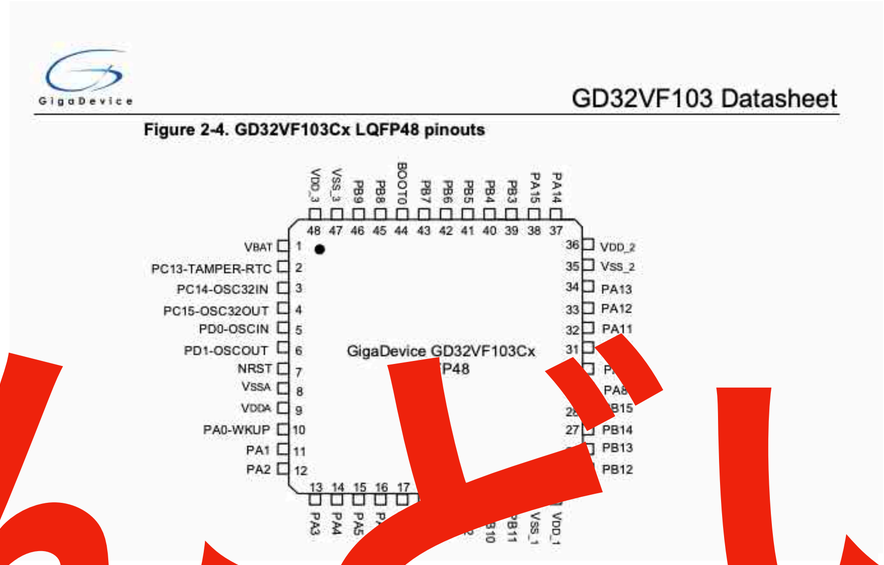
https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pd

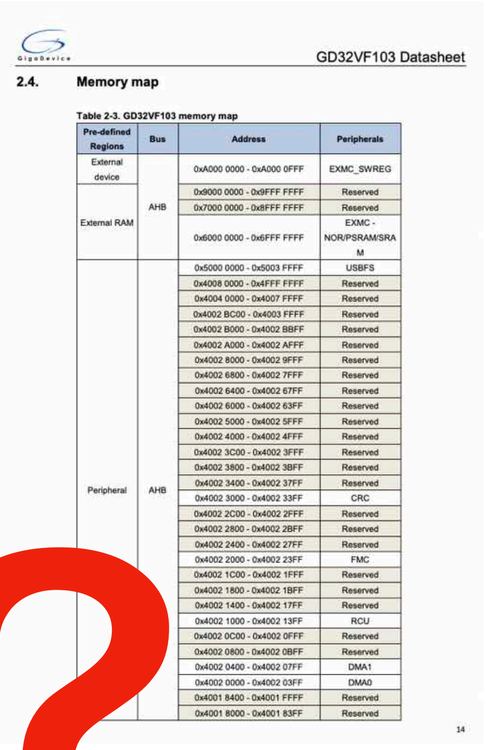
作り方は、CPUの**データシート**に書いてある

GigaDevice Semiconductor Inc.

GD32VF103
RISC-V 32-bit MCU

Datasheet





しんどい？

3.4. Bo

At startup, boot pins are used to select one of three boot options:

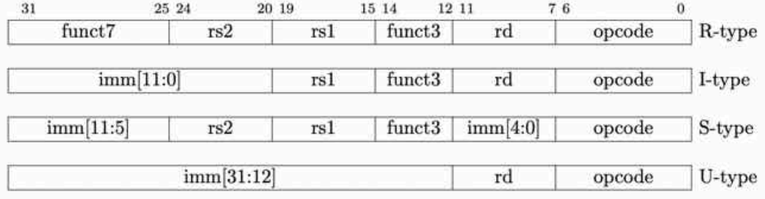
- Boot from main flash memory (default)
- Boot from system memory
- Boot from on-chip SRAM

The boot loader is located in the internal boot ROM memory (system memory). It is used to reprogram the Flash memory by using USART0 (PA9 and PA10), USART1 (PD5 and PD6), USBFS in device mode (PA9, PA11 and PA12). It also can be used to transfer and update the Flash memory code, the data and the vector table sections.

2.2 Base Instruction Formats

In the base ISA, there are four core instruction formats (R/I/S/U), as shown in Figure 2.2. All are a fixed 32 bits in length and must be aligned on a four-byte boundary in memory. An instruction address misaligned exception is generated on a taken branch or unconditional jump if the target address is not four-byte aligned. No instruction fetch misaligned exception is generated for a conditional branch that is not taken.

The alignment constraint for base ISA instructions is relaxed to a two-byte boundary when instruction extensions with 16-bit lengths or other odd multiples of 16-bit lengths are added.



file:///Users/fukuno/Downloads/GD32VF103_Datasheet_Rev1.0.pdf

https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pd

しんどいことをやってくれる
ソフトウェア

それが、OS（基本ソフトウェア）

！

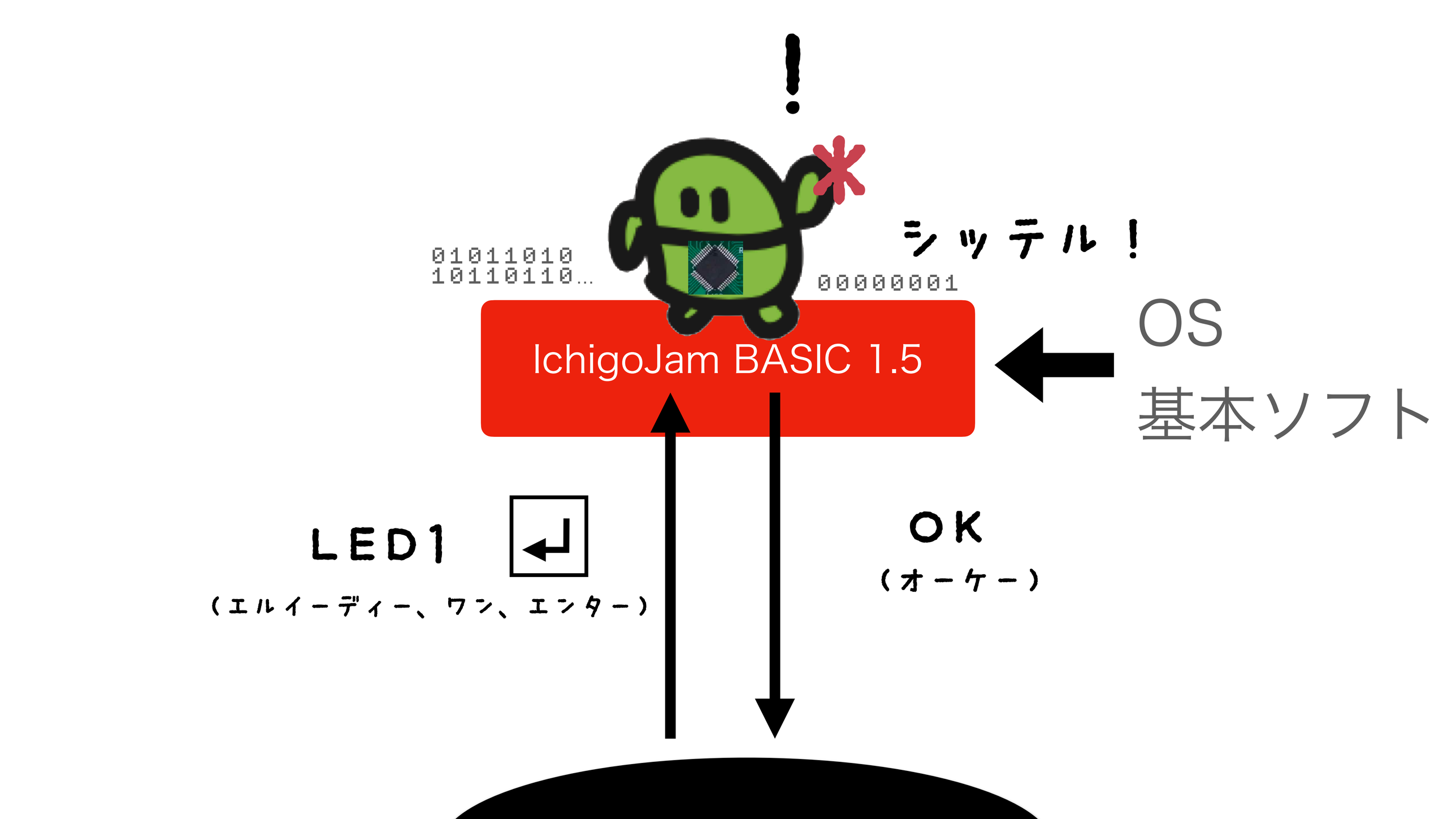
シッテル！

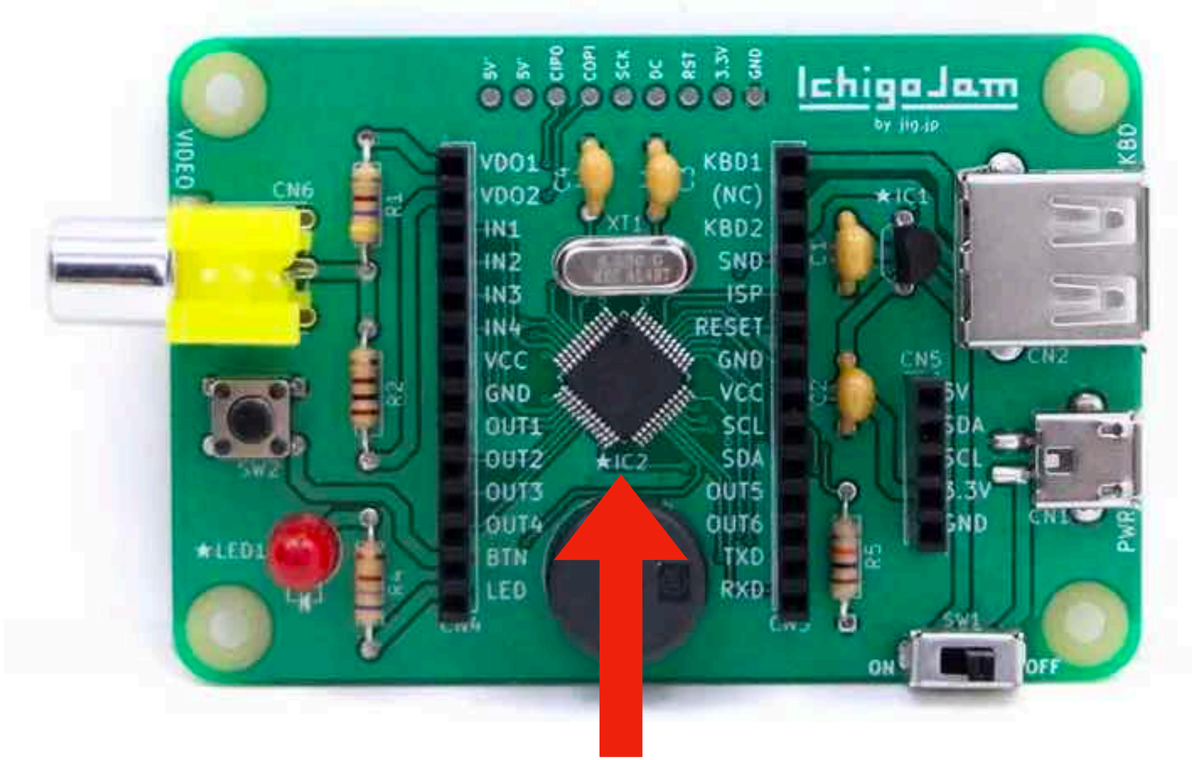
LED1

（エルイーディー、ワン、エンター）

OK

（オーケー）


！
01011010
10110110…
シッテル！
00000001
IchigoJam BASIC 1.5
OS
基本ソフト
LED1
（エルイーディー、ワン、エンター）
OK
（オーケー）



OS「IchigoJam BASIC 1.5」をインストール（書き込み）済み

コンピューターの仕組み

応用ソフトウェア アプリ	IchigoJamの プログラム	Blenderとか	iPhoneアプリ	Android アプリ	SWITCHの ゲーム
基本ソフトウェア OS	IchigoJam BASIC	macOS	iOS	Android	SWITCHのOS
ハードウェア ハード	IchigoJam R	Mac	iPhone	iPhone以外の スマホ	SWITCH

なぜ遅いか？

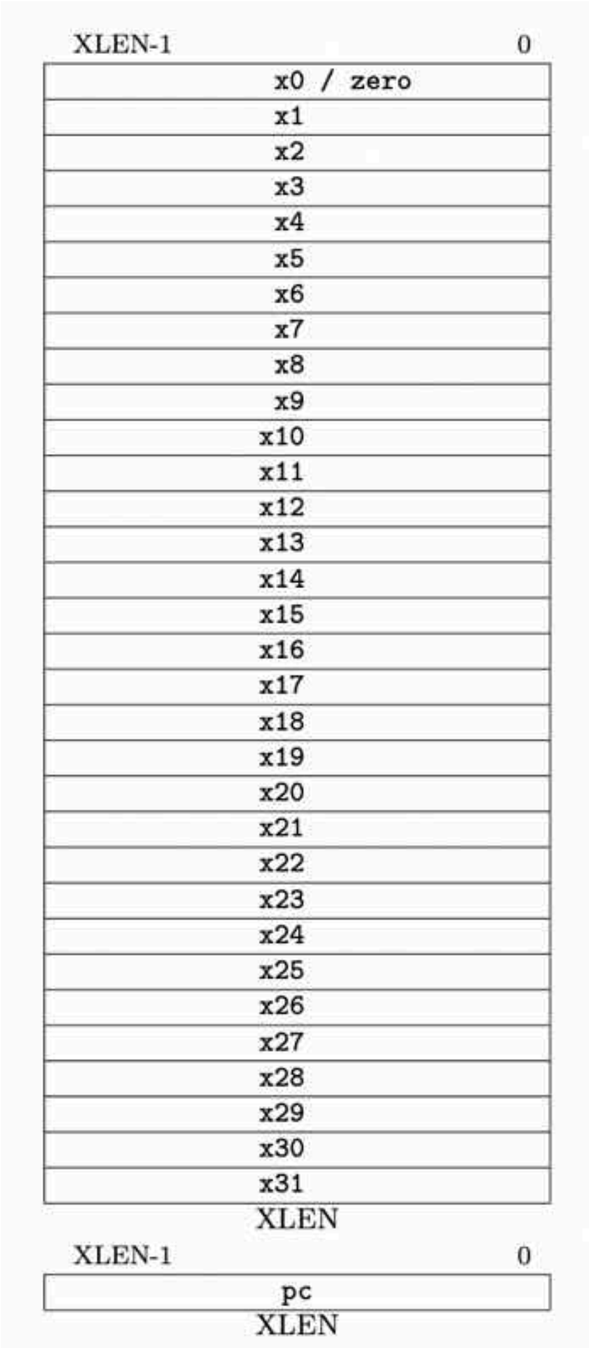
なぜ遅いか？

OSが、L、E、D、1 を一文字ずつ解釈して、翻訳しているから

IchigoJamの本気を
見てみよう！

マシン語はじめのいっぽ

「1」足そう



GD32VF103 (32bit RISC-V)

32コの**レジスタ**を持つ

レジスタ＝最も高速な一次記憶

IchigoJamのAとかの変数的なもの

R0～R31 と表記する

https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

レジスタR10に1を足すには？

Integer Register-Immediate Operations

These integer register-immediate operations are encoded in the CI format and perform operations on any non-`x0` integer register and a 6-bit immediate. The immediate cannot be zero.

15 13	12	11 7	6 2	1 0
funct3	imm[5]	rd/rs1	imm[4:0]	op
3	1	5	5	2
C.ADDI	nzimm[5]	dest	nzimm[4:0]	C1
C.ADDIW	imm[5]	dest≠0	imm[4:0]	C1
C.ADDI16SP	nzimm[9]	2	nzimm[4|6|8:7|5]	C1

C.ADDI adds the non-zero sign-extended 6-bit immediate to the value in register *rd* then writes the result to *rd*. C.ADDI expands into `addi rd, rd, nzimm[5:0]`.

https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

→0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 ね！

2進法知ってる？

2進法

0と1の2種類による数の表現法

→よくある間違い「2進数」
2進数という数なわけではないよ

10進法	2進法
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

1000まで記憶できる指二進法

データ量の単位 bit（ビット）は
2進法の桁数

（つまり、片手で5bit、両手で10bit）

1　0

最小のデータはあるかないかの1bit

バイト　　ビット

1byte = 8bit

0〜255の256種類の数を表せる（2の8乗）

英数1文字程度

10進法	2進法	16進法
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

	10進法へ	10進法から
2進法	?`1111	?BIN$(15)
16進法	?#F	?HEX$(15)

100は2進法で表現すると？　16進法のFFは10進法で表現すると？

レジスタR10に1を足すには？

Integer Register-Immediate Operations

These integer register-immediate operations are encoded in the CI format and perform operations on any non-`x0` integer register and a 6-bit immediate. The immediate cannot be zero.

15 13	12	11 7	6 2	1 0
funct3	imm[5]	rd/rs1	imm[4:0]	op
3	1	5	5	2
C.ADDI	nzimm[5]	dest	nzimm[4:0]	C1
C.ADDIW	imm[5]	dest≠0	imm[4:0]	C1
C.ADDI16SP	nzimm[9]	2	nzimm[4|6|8:7|5]	C1

C.ADDI adds the non-zero sign-extended 6-bit immediate to the value in register *rd* then writes the result to *rd*. C.ADDI expands into `addi rd, rd, nzimm[5:0]`.

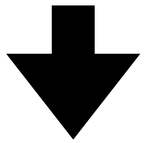
https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

5 5

→0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 ね！

5　　5

→00000101000001 01 ね！



```
POKE#700,5,5
```

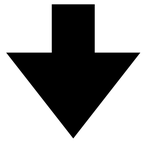
＊POKE = CPUのメモリにセットするコマンド

```
?USR(#700,1)
```

＊USR = CPUに次実行する場所を伝えるコマンド

暴走！

戻っておいでというマシン語



```
POKE#700,5,5,130,128
```

＊POKE = CPUのメモリにセットするコマンド

```
?USR(#700,1)
```

＊USR = CPUに次実行する場所を伝えるコマンド

本気を測ろう

R11=0

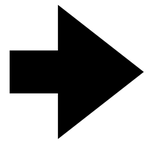
R11+=1

R10-=1

IF R10 GOTO -2

R10=R11

RET



0100010110000001

0000010110000101

0001010101111101

1111110101110101

1000010100101110

1000000010000010

```
10  POKE#700,129,69,133,
5,125,21,117,253,46,133
,130,128
20  CLT
30  A=USR(#700,100000)
40  ?TICK()/6
```

```
10  POKE#700,129,69,133,
5,125,21,117,253,46,133
,130,128
20  CLT
30  A=USR(#700,100000)
40  ?TICK()
```

まだ0!?

```
10  POKE#700,129,69,133,
5,125,21,117,253,46,133
,130,128
20  CLT
30  A=USR(#700,100000)
40  ?TICK(1)
```

TICKの分解能を261倍にするオプション

＊リファレンス参照

12/(60*261)=0.00077秒

1/0.00077=1300

1万回ループなので、1,300万ループ/秒

```
R11=0
R11+=1
R10-=1
IF R10 GOTO -2
R10=R11
RET
```

計算が2コ
判定が1コ
GOTOが1コ

→4コ計算

1300万x4計算 = 計算5200万回/秒

ちょっと足りない分は画面表示やキーボード処理など裏タスク

マシン語はしんどいけど、IchigoJam BASICの**3千倍速い**！

	IchigoJam BASIC	マシン語
書きやすさ	○	×
安全	○	×
速度	×	○
ループ/秒	4,500回/秒	1,300万回/秒

ふくっちは、
こんなしんどいことをやって
IchigoJam BASICを作ったのか！？