

# 情報科学特別講義A 第5回 成果発表

## Webブラウザ上でC言語を動かすための仮想マシンの試作

杉田基樹

# 仮想マシンの概要

# 実装した仮想マシンのコンセプト

- Webブラウザ上で実行可能にする
  - ✓ どこでも動く
  - ✓ cloneなどをせずにすぐ試せる
- C言語を動かすことを想定する
  - ✓ 知っている人が多いので伝わりやすい

# 実装に使用した言語

- TypeScript
  - 静的型付けを取り入れたJavaScript
  - ブラウザで動く唯一(?)の言語
  - 主な言語仕様
    - 型・データ構造
      - 数値(number)
      - 文字列(string)
      - 真偽値(boolean)
      - 配列(Array)
      - オブジェクト(Object)
        - クラス・インターフェース
      - 列挙型(enum)
        - 非推奨
      - ユニオン型

```
let value: string | number;  
value = 1; // OK  
value = "foo"; // OK  
const array: (string | number)[] = [1, "foo"];
```



<https://www.typescriptlang.org/branding/>

# 実装した機能

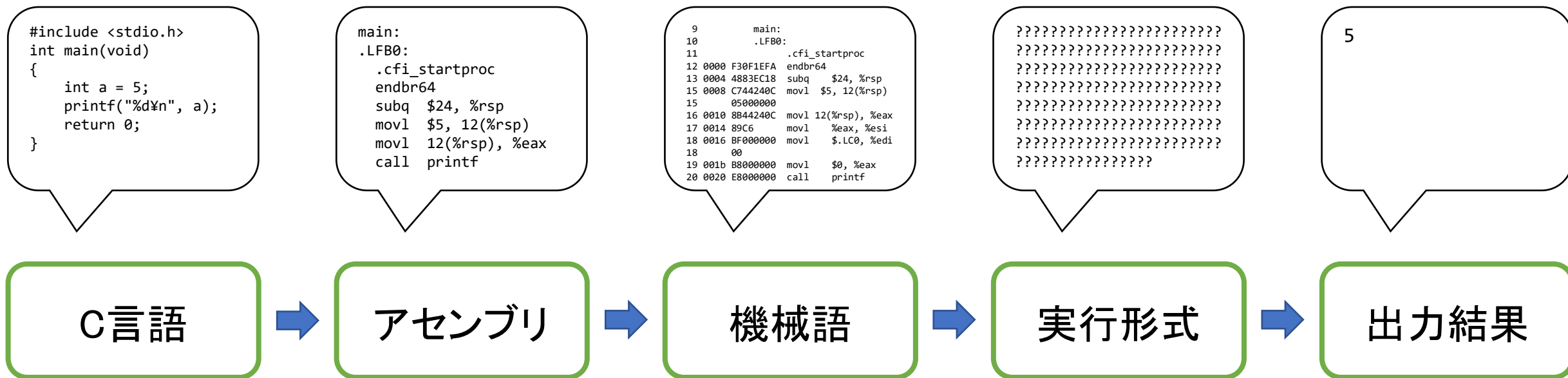
- 演算
  - 算術
  - 比較
- 変数
  - 型
  - アドレス
- 関数
  - ローカル変数
- 配列
- ポインタ

# 設計・実装の工夫点

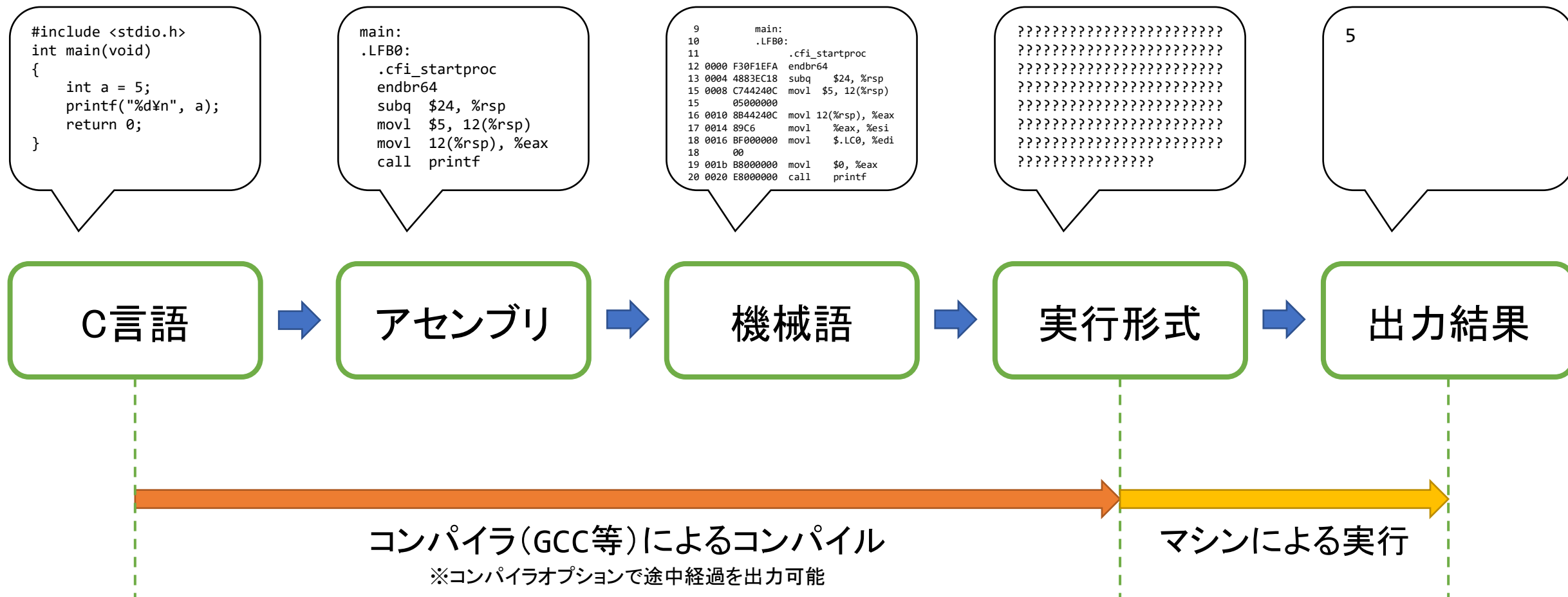
- C言語を動かすための工夫
  - 文法・型などをなるべくCに近づける
  - TypeScriptのオブジェクト指向的な側面を利用して、メモリ管理を簡単にする
- 高速化のための工夫
  - アセンブリから直接実行せず、独自の機械語を挟む
  - 機械語を文字列ではなくTypeScriptのデータ構造で表現する

次のスライドで補足

# C言語のプログラムを実行するときの流れ

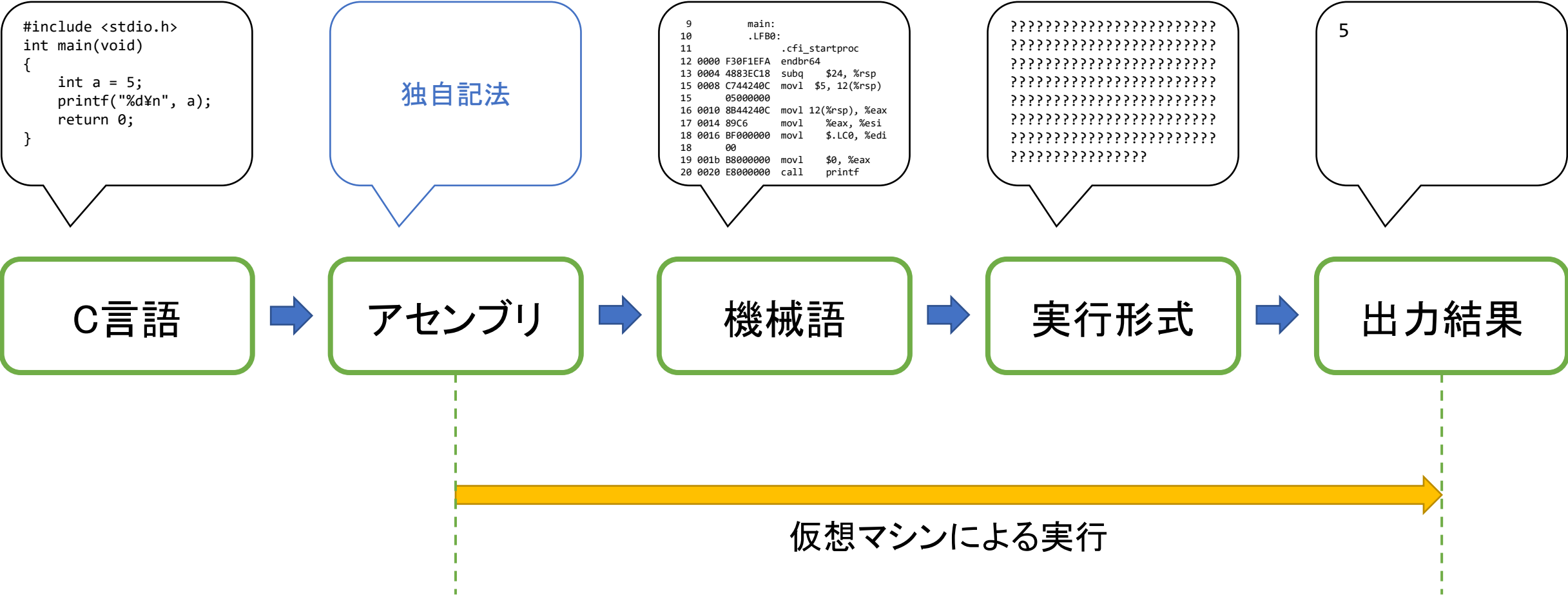


# C言語のプログラムを実行するときの流れ

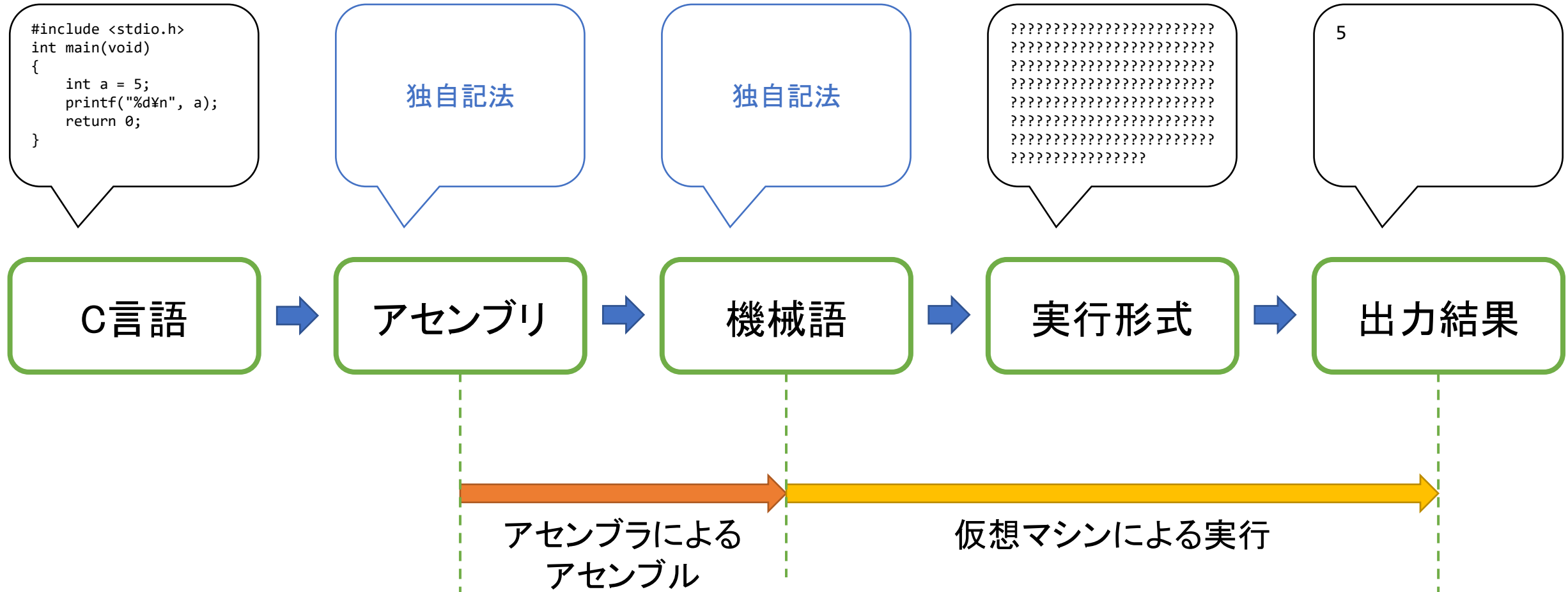




# 第4回までの実装



# 今回の実装



# アセンブラの設計

# アセンブリの記述例

TWICE:

```
declareLocal value int
setLocal value
getLocal value
push 2
mul
return
```

MAIN:

```
declareLocal array[2] int
push 3
setLocal array[0]
push 4
setLocal array[1]

getLocal array[0]
call TWICE
call TWICE
getLocal array[1]
call TWICE
add
print
```

- 適当な計算を行うプログラム
  - $3 * 2 * 2 + 4 * 2$
  - 出力は20

# アセンブラによる前処理

TWICE:

```
declareLocal value int
setLocal value
getLocal value
push 2
mul
return
```

MAIN:

```
declareLocal array[2] int
push 3
setLocal array[0]
push 4
setLocal array[1]

getLocal array[0]
call TWICE
call TWICE
getLocal array[1]
call TWICE
add
print
```


- 空行を消す
- ラベルを辞書登録
  - TWICE -> 0
  - MAIN -> 6
- 文字列を削ぎ落とす
  - 変数名
    - value -> 0
    - array -> 1
  - 型名
    - int -> 0

```
00 declareLocal 0 0
01 setLocal 0
02 getLocal 0
03 push 2
04 mul
05 return
06 declareLocal 1[2] 0
07 push 3
08 setLocal 1[0]
09 push 4
10 setLocal 1[1]
11 getLocal 1[0]
12 call 0
13 call 0
14 getLocal 1[1]
15 call 0
16 add
17 print
```

# 機械語への変換

- 辞書を用いて命令を数値化

00	declareLocal	0	0
01	setLocal	0	
02	getLocal	0	
03	push	2	
04	mul		
05	return		
06	declareLocal	1[2]	0
07	push	3	
08	setLocal	1[0]	
09	push	4	
10	setLocal	1[1]	
11	getLocal	1[0]	
12	call	0	
13	call	0	
14	getLocal	1[1]	
15	call	0	
16	add		
17	print		



00	21	0	0
01	22	0	
02	23	0	
03	2	2	
04	5		
05	25		
06	21	1	2
07	2	3	
08	22	1	0
09	2	4	
10	22	1	1
11	23	1	0
12	24	0	
13	24	0	
14	23	1	1
15	24	0	
16	3		
17	0		

# 仮想マシンの設計

# 仮想マシンが入力として受け取る機械語

- 1命令をオブジェクト化し、配列に入れて命令列を作る
- 文字列はできる限り排除する

```
const instruction = [  
  { methodId: 2, arguments: [10] },  
  { methodId: 2, arguments: [5] },  
  { methodId: 3, arguments: [] },  
  { methodId: 0, arguments: [] },  
]
```

VMに入力される機械語の例



```
push 10  
push 5  
add  
print
```

変換前のアセンブリ

```
private methods = [  
  /* No.00 */ this._print,  
  /* No.01 */ this._pop,  
  /* No.02 */ this._push,  
  /* No.03 */ this._add,  
  /* No.04 */ this._sub,  
  /* No.05 */ this._mul,  
  /* No.06 */ this._div,  
  /* No.07 */ this._mod,  
  /* No.08 */ this._eq,  
  /* No.09 */ this._ne,  
  /* No.10 */ this._gt,  
  /* No.11 */ this._ge,  
  /* No.12 */ this._lt,  
  .....  
]
```

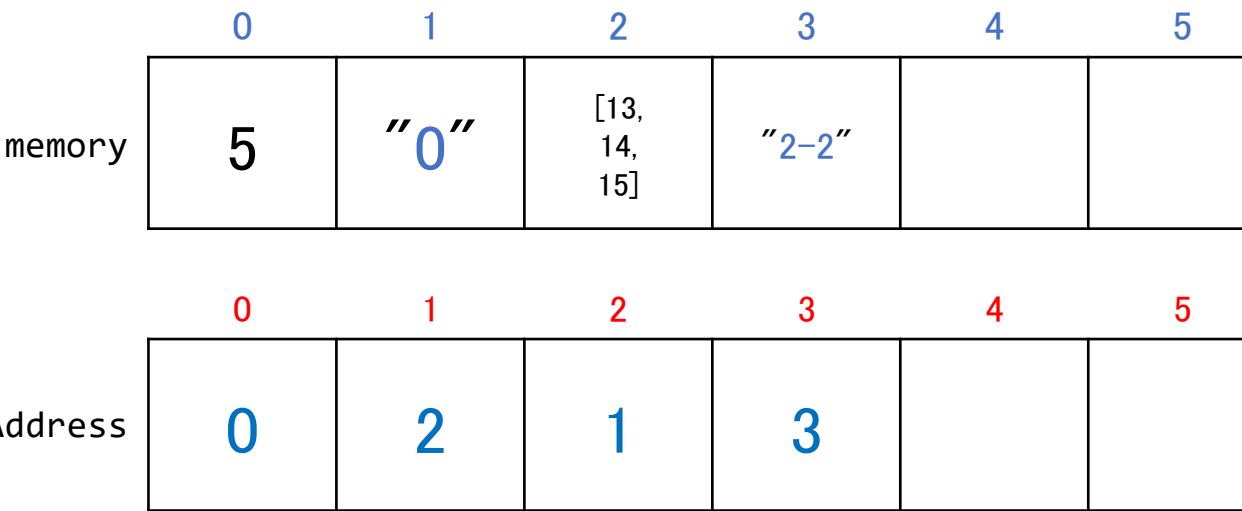
内部に持つメソッド一覧表



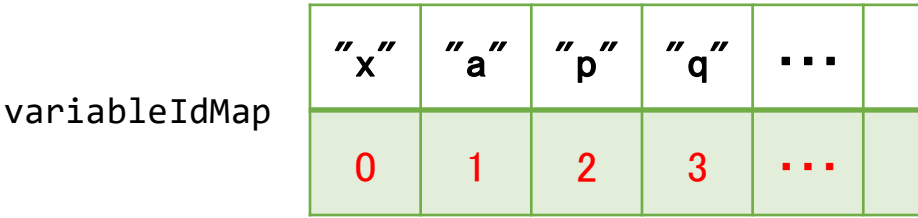
# 変数の管理方法

- 1本のメモリにすべてを入れる

```
int x;  
int a[3];  
int *p, *q;  
  
x = 5;  
a[0] = 13;  
a[1] = 14;  
a[2] = 15;  
  
p = &x;  
q = &a[2]
```



※アセンブラ



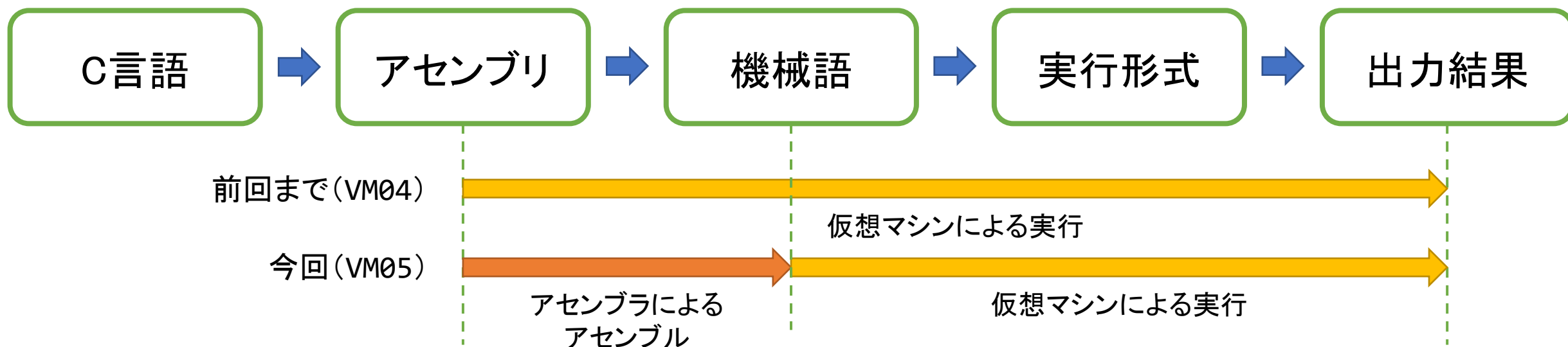
# 速度比較

# これまでに作ったVMと比較

- 比較対象
  - 第4回までで作成したVM
  - 「TSで実装したC言語用VM」とコンセプトは同じ
  - アセンブルを実質行わず、VMが文字列を読みながら実行
- ベンチマーク
  - フィボナッチ数列の第1項からN項までを出力するプログラム
  - 再帰呼出しによる実装
- 実行環境
  - ソフトウェア
    - Windows11
    - Google Chrome
  - ハードウェア
    - 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
    - メモリ 16.0 GB

# 実験の詳細

- 出力するフィボナッチ数列の項数Nを 5 から 30 まで 5 刻みで増加させ、それぞれでかかった時間を計測する
- 同じ条件で5回試し、そのうちの最小値を記録とする
- 平等性を担保するため、アセンブリの入力から実行結果の出力までを測る



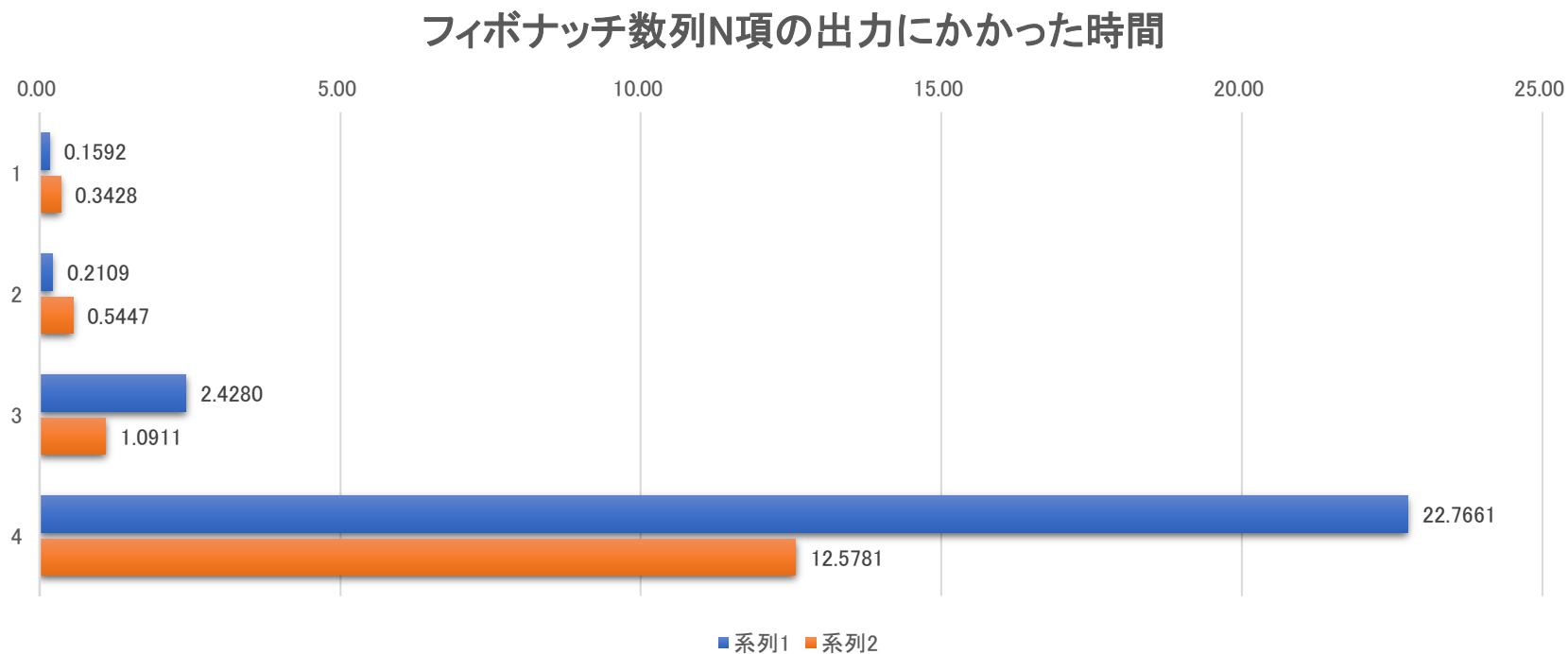
# 実験結果

- Nが15を超えてからは圧倒的に高速になった

N(回)	5	10	15	20	25	30
VM04(ms)	0.1592	0.2109	2.4280	22.7661	206.9048	2481.1108
VM05(ms)	0.3428	0.5447	1.0911	12.5781	165.9058	1871.9617
VM05/VM04(%)	215	258	45	55	80	75

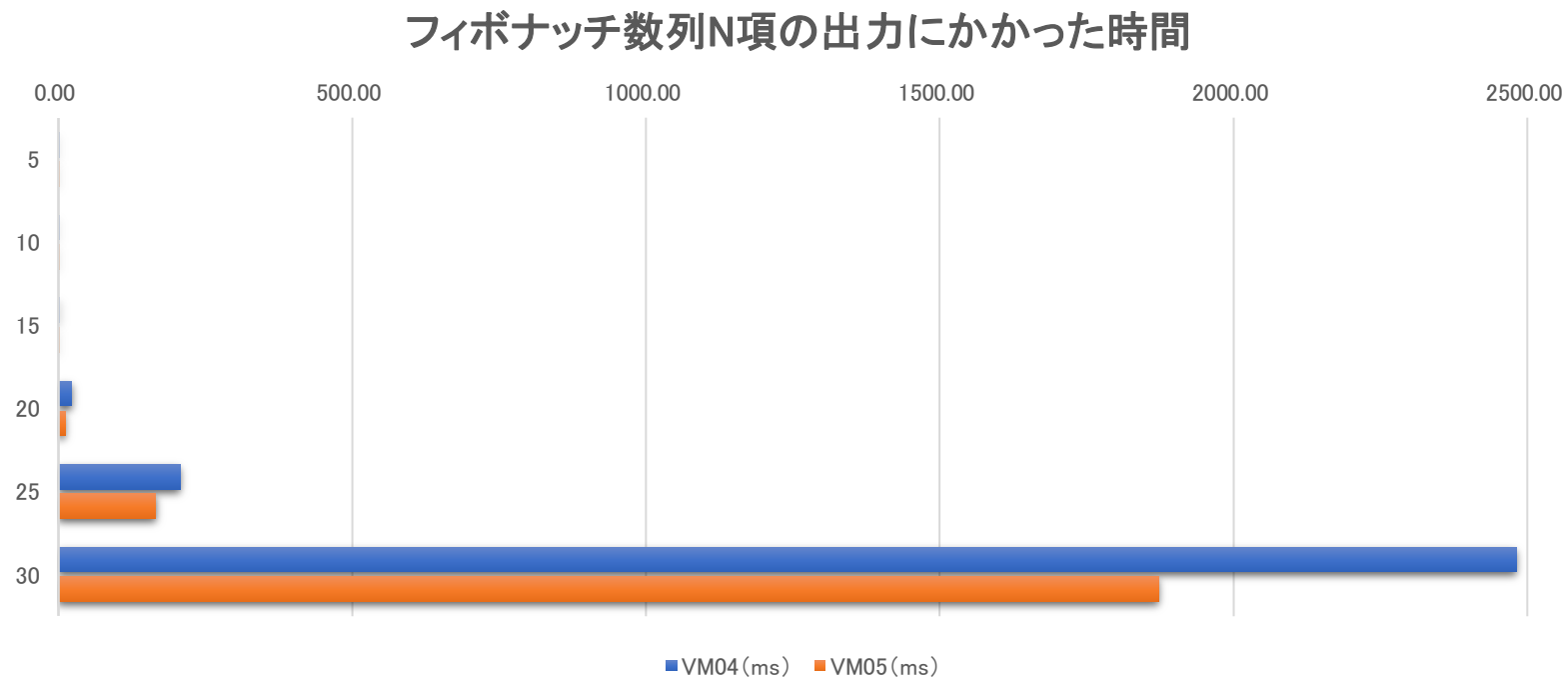
# 実験結果

- Nが5～20までのグラフ



# 実験結果

- 結果全体のグラフ



# 結果の考察

- なぜ高速化したか
  - VMが文字列の解釈をしなくなったため
  - TSは文字列に対する処理がそこまで速くない
- なぜNが10以下では遅かったか
  - アセンブラによる変換のオーバーヘッドがあったため
  - 変数のSet・GetのほかにDeclareを設けたため



# まとめ

# まとめ

- Webブラウザ上でC言語を動かすための仮想マシンを試作した
- 今後の展望
  - 実装されていないデータ構造などを仕上げる
    - 構造体
    - 列挙
    - 関数ポインタ
    - 標準ライブラリ
  - Cからアセンブリへの変換器(コンパイラ?)を作る