ミクロン・シミュレータ

マニュアル

目次

ミクロン・シミュレータについて

- ミクロン・シミュレータとは、コンピュータが機械語プログラムを実行している様子を見るための、超小型CPUシミュレータです。
- 大まかな操作の流れは、(1) 機械語のプログラムやデータをメモリに記述していく、(2) 実行ボタンを押す、です。0番地の命令から順にプログラムが実行されます。
- 実際の機械語は0と1が並んだ2進数で表されますが、読んだり書いたりが大変なので、このシミュレータでは16進数で表示・入力します。

各機能の説明

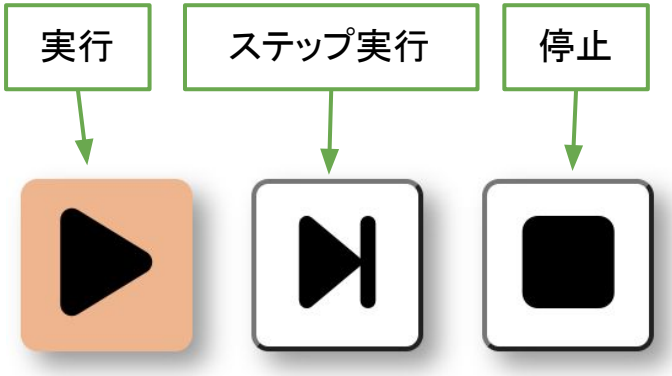
- 全体像
- 実行/ステップ実行/停止ボタン
- レジスタ/フラグ
- メモリ
- アップロード/ダウンロードボタン
- メモリ編集ボタン

各機能の説明　~全体像~


実行/ステップ実行/停止ボタン
ミクロン・シミュレータ
レジスタ/フラグ
実行速度 0.125秒／1命令
レジスタ/フラグ
A B C IP Z C
00 00 00 00 F F
メモリ
メモリ
アップロード/
ダウンロード
ボタン
メモリ編集ボタン
メモリ編集
0x00番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x30番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x50番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x60番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x70番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x80番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x90番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xA0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xB0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xC0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xD0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xF0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

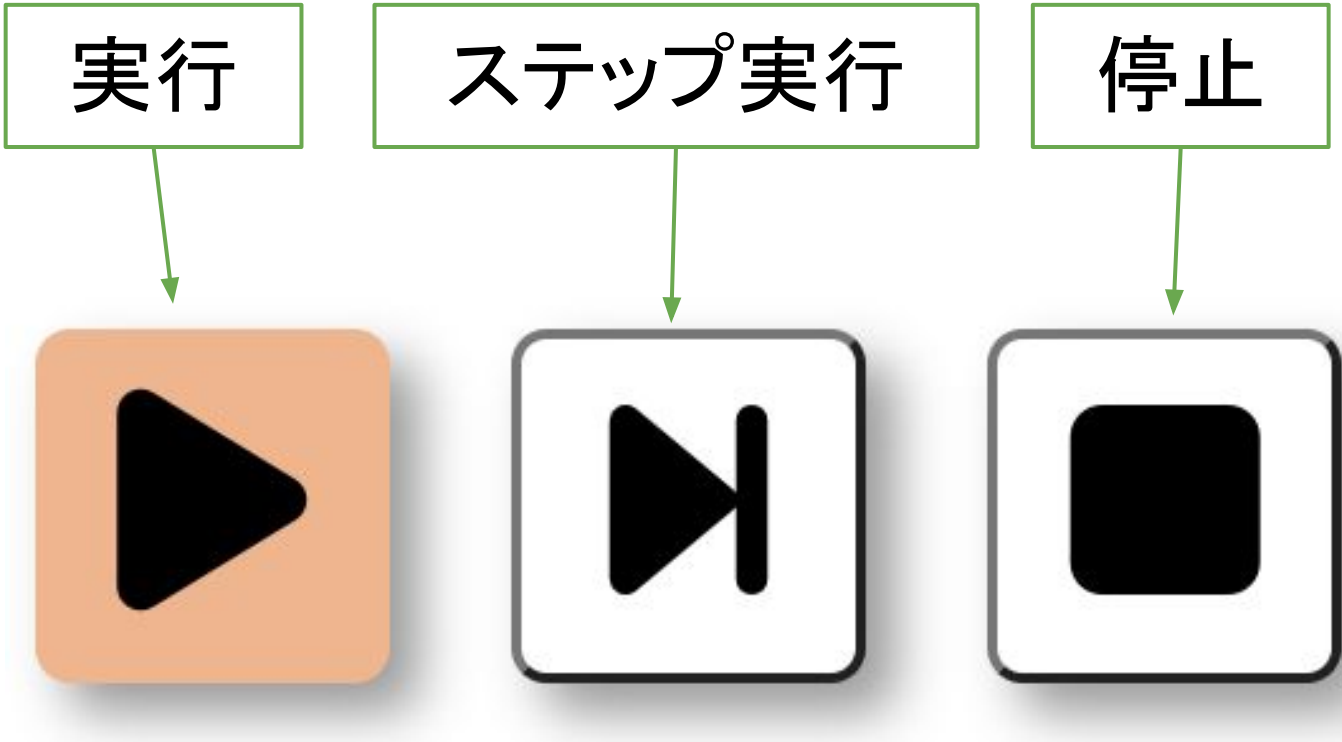

実行/ステップ実行/停止ボタン
ミクロン・シミュレータ
レジスタ/フラグ
実行速度 0.125秒／1命令
レジスタ/フラグ
A B C IP Z C
00 00 00 00 F F
メモリ
メモリ
0x00番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x30番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x50番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x60番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x70番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x80番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x90番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xA0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xB0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xC0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xD0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xF0番地 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
アップロード/
ダウンロード
ボタン
メモリ編集ボタン
メモリ編集

各機能の説明　~実行/ステップ実行/停止ボタン~



- 実行
  - 最初から最後まで実行します
- ステップ実行
  - 1命令ずつ実行します
  - 実行中に選択すると現在処理している命令の次の命令を処理して止まります
  - 止まった後に再度ステップ実行を押すと次の 1命令を実行します
- 停止
  - 1回だけ押すと停止し、その後に実行ボタンを押すと続きから処理されます
  - 2回押すとレジスタやフラグは初期化され、その後に実行ボタンを押すと初めから処理されます

- 1命令の実行に掛かる時間を表します
- 4段階で変更ができ、秒数が大きくなるほどプログラム全体の実行時間が長くなります



実行速度 0.125秒／1命令

各機能の説明　~レジスタ/フラグ~

レジスタ/フラグ

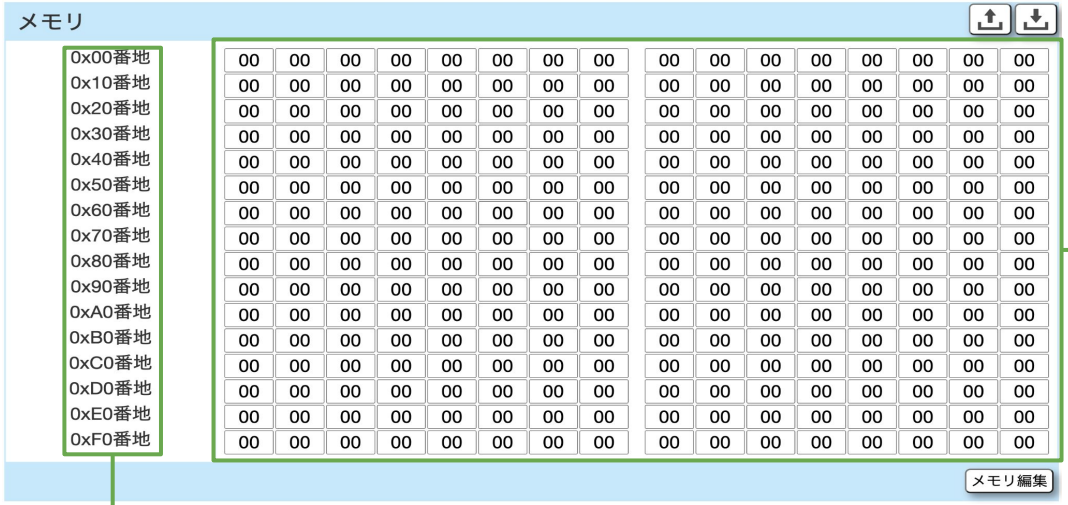
A	B	C	IP	Z	C
00	00	00	00	F	F

- A～IPがレジスタで、ZとCがフラグです
- 各レジスタに格納されている値や、フラグの状態（ TrueではT、FalseではF）が表示されます
- A～CはJavaなどでいう変数のようなもので、自由に値を書き換えることができます
- IPは現在実行しているメモリの番地が格納されるレジスタで、（ジャンプ命令以外で）値を変更することはできません
- Z (zeroフラグ) は計算結果の下位8ビットが0の時や、値同士を比較した際等しかった時に Tになります
- C (carryフラグ) は計算結果が256以上（桁上がり）の時や負数（桁借り）の時、値同士を比較した際「<」が成り立った時にTになります

レジスタ/フラグ

A	B	C	IP	Z	C
00	00	00	00	F	F

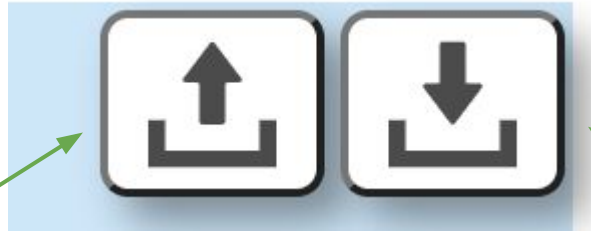
各機能の説明　~メモリ~



- メモリマップ
  - 機械語のプログラムやデータを記述する場所です
  - 記述する際は16進数を用います
  - メモリ選択時にEnterキーを押すと次のメモリが選択されます

各行の先頭のメモリの番地

各機能の説明　~アップロード/ダウンロードボタン~



アップロードボタン
- ファイルに保存されているメモリ内容をアップロードします
- 使用可能なファイルの拡張子は以下の 2つです
  - .hex
  - .txt

ダウンロードボタン
- メモリマップの全内容をファイルに保存（ダウンロード）します
- program.hexというファイル名で保存されますが、ダウンロード時にファイル名と保存先を指定したい場合は、ブラウザの設定を変更してください（Firefoxの設定変更方法）

各機能の説明 ~メモリ操作~

指定した番地以降のメモリを，
左または右に移動，またはゼロクリアします。

例えば番地入力欄に「E0」と記入して「左に移動」ボタンを押すと，0xE1番地の値が0xE0番地に，0xE2番地の値が0xE1番地に，…，0xFF番地の値が0xFE番地に移動します。

「E0」と記入して「ゼロクリア」ボタンを押すと，0xE0～0xFF番地の値が00になります。全メモリを00にしたい場合は「00番地以降」を「ゼロクリア」してください。

命令セット

16進数	命令名	意味	引数
00	hlt	終了命令	なし
01	mov_imm	レジスタに即値を格納	レジスタ、即値（00～ff）
02	mov_reg	レジスタ1にレジスタ2の値を格納	レジスタ1、レジスタ2
03	add	レジスタ1にレジスタ2＋レジスタ3の結果を格納	レジスタ1、レジスタ2、レジスタ3
04	sub	レジスタ1にレジスタ2ーレジスタ3の結果を格納	レジスタ1、レジスタ2、レジスタ3
05	cmp	レジスタ1とレジスタ2を比較	レジスタ1、レジスタ2
06	b	指定された番地にジャンプ	番地（00～ff）

命令セット

16進数	命令名	意味	引数
07	bgt	「>」であったとき指定された番地にジャンプ	番地（00～ff）
08	bge	「>=」であったとき指定された番地にジャンプ	番地（00～ff）
09	beq	「==」であったとき指定された番地にジャンプ	番地（00～ff）
0a	bne	「!=」であったとき指定された番地にジャンプ	番地（00～ff）
0b	ble	「<=」であったとき指定された番地にジャンプ	番地（00～ff）

命令セット

16進数	命令名	意味	引数
0c	blt	「<」であったとき指定された番地にジャンプ	番地（00～ff）
0d	str	レジスタ2に格納された番地にレジスタ1の値をコピー	レジスタ1、レジスタ2
0e	ldr	レジスタ1にレジスタ2に格納された番地の値をコピー	レジスタ1、レジスタ2

レジスタ番号

16進数	レジスタ
01	A
02	B
03	C

各命令の説明

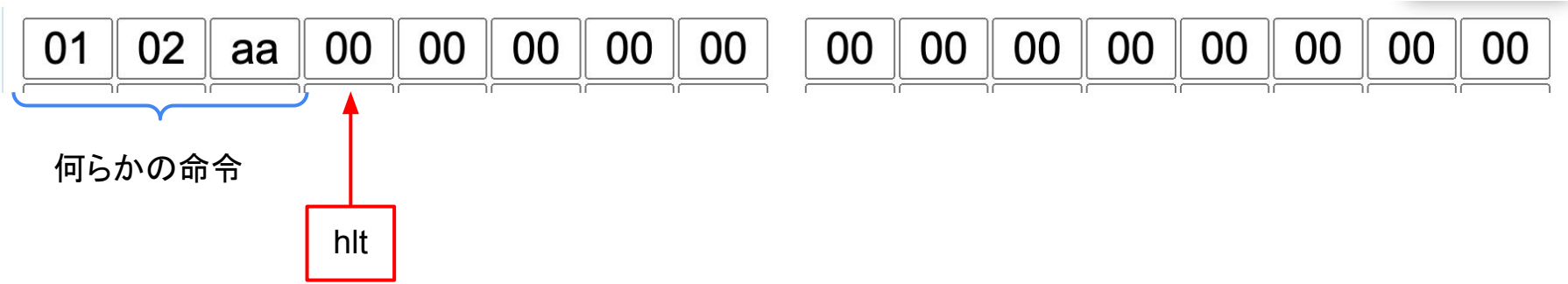
- hlt
- mov_imm
- mov_reg
- add
- sub
- cmp
- b
- bgt
- bge, beq, bne, ble, blt
- str
- ldr

各命令の説明 ~hlt~

16進数	命令名	意味	引数
00	hlt	終了命令	なし

- プログラムを終了します
- プログラムの最後に必ず使用してください

使用例



各命令の説明 ~mov_imm~

16進数	命令名	意味	引数
01	mov_imm	レジスタに即値を格納	レジスタ、即値（00～ff）

- 指定したレジスタに即値（好きな値）を格納します
- レジスタはレジスタ番号で指定してください
- 即値として使用できるのは00～ffです
- 命令番号（01）、レジスタ番号、即値の順に書いてください

使用例

01	02	aa	00	00	00	00	00	00	00	00	00	00	00	00	00

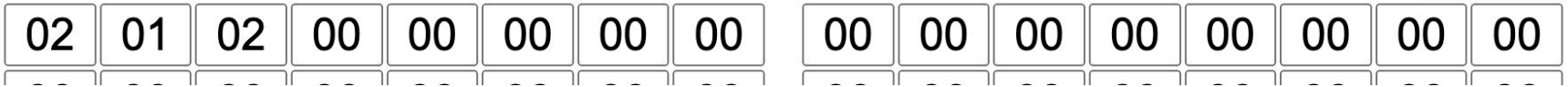
この場合は、レジスタBにaa（10進数では170）が格納されます

各命令の説明 ~mov_reg~

16進数	命令名	意味	引数
02	mov_reg	レジスタ1にレジスタ2の値を格納	レジスタ1、レジスタ2

- レジスタ1にレジスタ2に格納されている値をコピーします
- レジスタはレジスタ番号で指定してください
- 命令番号（02）、格納先レジスタ番号、コピー元レジスタ番号の順に書いてください

使用例



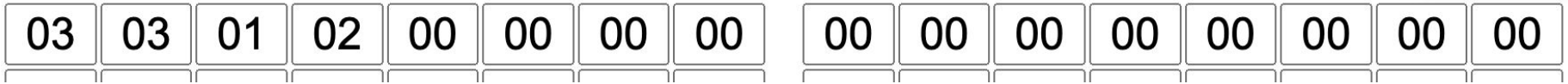
この場合は、レジスタBに格納されていた値がレジスタAにコピーされます

各命令の説明　~add~

16進数	命令名	意味	引数
03	add	レジスタ1にレジスタ2＋レジスタ3の結果を格納	レジスタ1、レジスタ2、レジスタ3

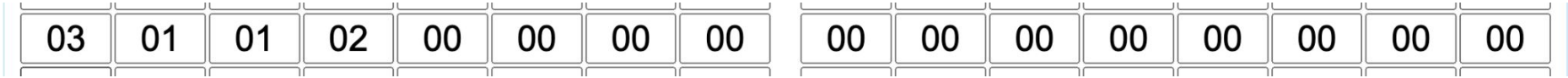
- レジスタ1にレジスタ2に格納されている値とレジスタ3に格納されている値の加算結果を格納します
- 計算後、レジスタ1のみ値が変わります。レジスタ2、レジスタ3は、レジスタ1と異なるレジスタであれば、値は変わりません
- レジスタはレジスタ番号で指定してください
- 命令番号（02）、結果の格納先レジスタ番号、和を計算したいレジスタその1、和を計算したいレジスタその2の順に書いてください

使用例1



この場合は、レジスタCにレジスタA＋レジスタBの計算結果が格納されます

使用例2



この場合は、レジスタAにレジスタAとレジスタBの計算結果が格納されます
Javaの「A += B」というコードと同じです

各命令の説明　~sub~

16進数	命令名	意味	引数
04	sub	レジスタ1にレジスタ2ーレジスタ3の結果を格納	レジスタ1、レジスタ2、レジスタ3

- addが減算になっただけなので、書き方などはaddと特に変わりません
- 具体的な使用方法はaddの使用例を参考にしてください

各命令の説明 ~cmp~

16進数	命令名	意味	引数
05	cmp	レジスタ1とレジスタ2を比較	レジスタ1、レジスタ2

- レジスタ1に格納されている値とレジスタ2に格納されている値を比較します
- レジスタ1はレジスタ2と比べて、大きい、小さい、等しい、等しくないというような判断をします（結果は Z, C フラグの値で表されます）
- この命令は条件付きのジャンプ命令の直前に使用することを想定しているため、単体で使用してもあまり意味はありません
- レジスタはレジスタ番号で指定してください

使用例

05	01	02	00	00	00	00	00

00	00	00	00	00	00	00	00

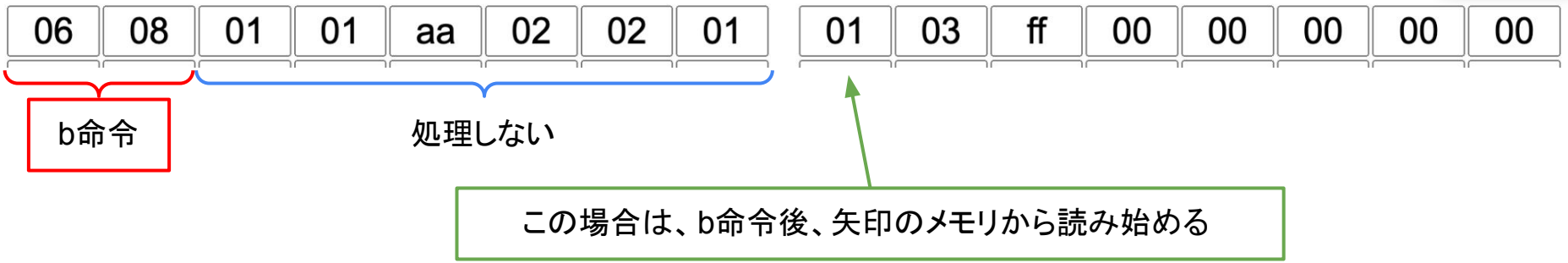
この場合は、レジスタAに格納されている値がレジスタBに格納されている値と比べて、
どうなのかを比較しています

各命令の説明　~b~

16進数	命令名	意味	引数
06	b	指定された番地にジャンプ	番地（00～ff）

- 指定された番地にジャンプします
- 番地として使用できるのは00～ffです

使用例



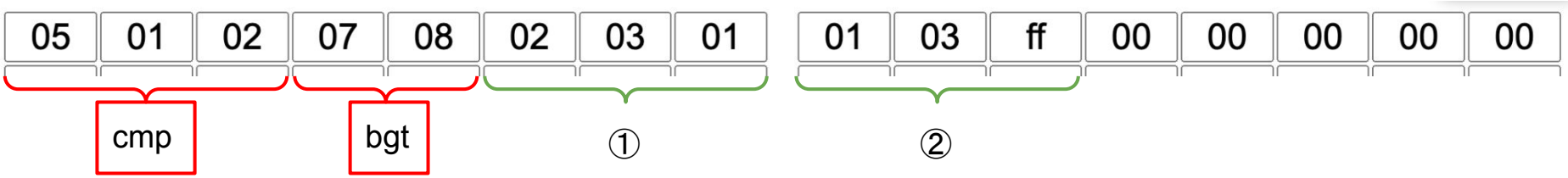
各命令の説明　~bgt~

16進数	命令名	意味	引数
07	bgt	「>」であったとき指定された番地にジャンプ	番地（00～ff）

- この命令を使用する際は、通常は1つ前の命令にcmpを使用してください
- cmpの結果が「>」であった時に指定された番地にジャンプします
- 番地として使用できるのは00～ffです

※bgt の正確な意味は「ZフラグもCフラグも F のときにジャンプ」です。同様に、beq の正確な意味は「Zフラグが T のときにジャンプ」、blt の正確な意味は「Cフラグが T のときにジャンプ」です。そのため、必ずしもcmpと組にする必要はなく、例えば beqを「subの結果が0だったときにジャンプ」する命令としても使えるし、bltを「addの結果桁上がりが生じたときにジャンプ」する命令として使用することもできます

使用例



この場合は、レジスタAとレジスタBに格納されている値同士を比較し、「 >」が成り立つ時は②にジャンプし、成り立たない時はジャンプせずに通常通り①から順に読みます

- レジスタA = 01　レジスタB = 02　　ジャンプしません
- レジスタA = 02　レジスタB = 02　　ジャンプしません
- レジスタA = 03　レジスタB = 02　　ジャンプします

各命令の説明 ~bge, beq, bne, ble, blt~

16進数	命令名	意味	引数
08	bge	「>=」であったとき指定された番地にジャンプ	番地（00～ff）
09	beq	「==」であったとき指定された番地にジャンプ	番地（00～ff）
0a	bne	「!=」であったとき指定された番地にジャンプ	番地（00～ff）
0b	ble	「<=」であったとき指定された番地にジャンプ	番地（00～ff）
0c	blt	「<」であったとき指定された番地にジャンプ	番地（00～ff）

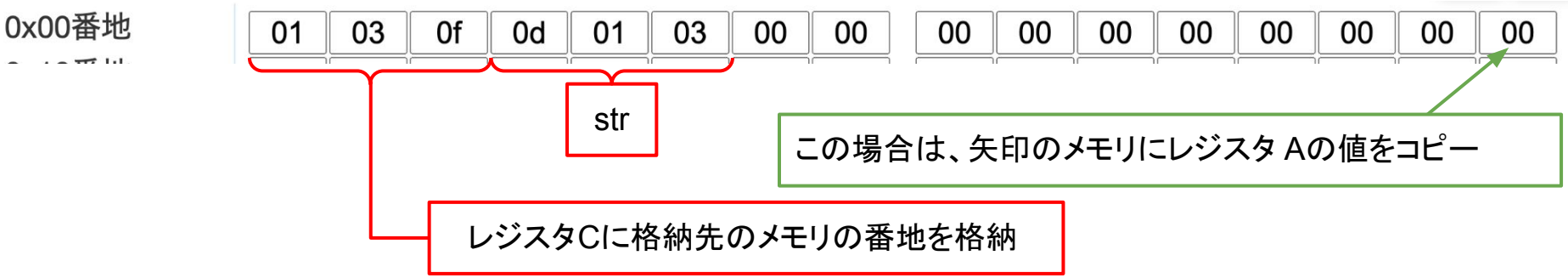
- bgtの条件が違うだけで使用方法はbgtと特に変わらないため、使用する際はbgtの使用例を参考にしてください

各命令の説明 ~str~

16進数	命令名	意味	引数
0d	str	レジスタ2に格納された番地にレジスタ1の値をコピー	レジスタ1、レジスタ2

- レジスタ2に格納された番地のメモリに、レジスタ1に格納されている値をコピーします
- 引数にはレジスタしか取れないため、strを使用する前に必ず格納先のメモリの番地をレジスタに格納してください

使用例


0x00番地
01
03
0f
0d
01
03
00
00
00
00
00
00
00
00
00
00
str
この場合は、矢印のメモリにレジスタ Aの値をコピー
レジスタCに格納先のメモリの番地を格納

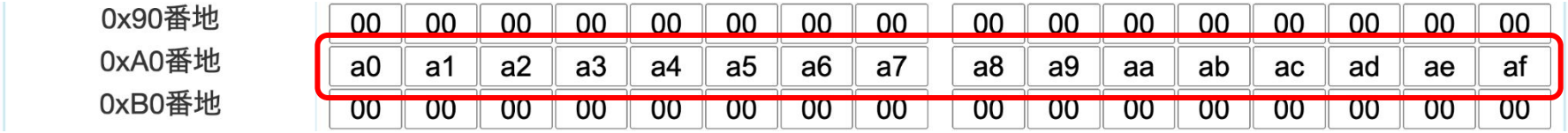
各命令の説明 ~ldr~

16進数	命令名	意味	引数
0e	ldr	レジスタ1にレジスタ2に格納された番地の値をコピー	レジスタ1、レジスタ2

- strの逆の動作で、指定したメモリの値をレジスタにコピーします
- 使用方法はstrと特に変わらないため、strの使用方法を参考にしてください

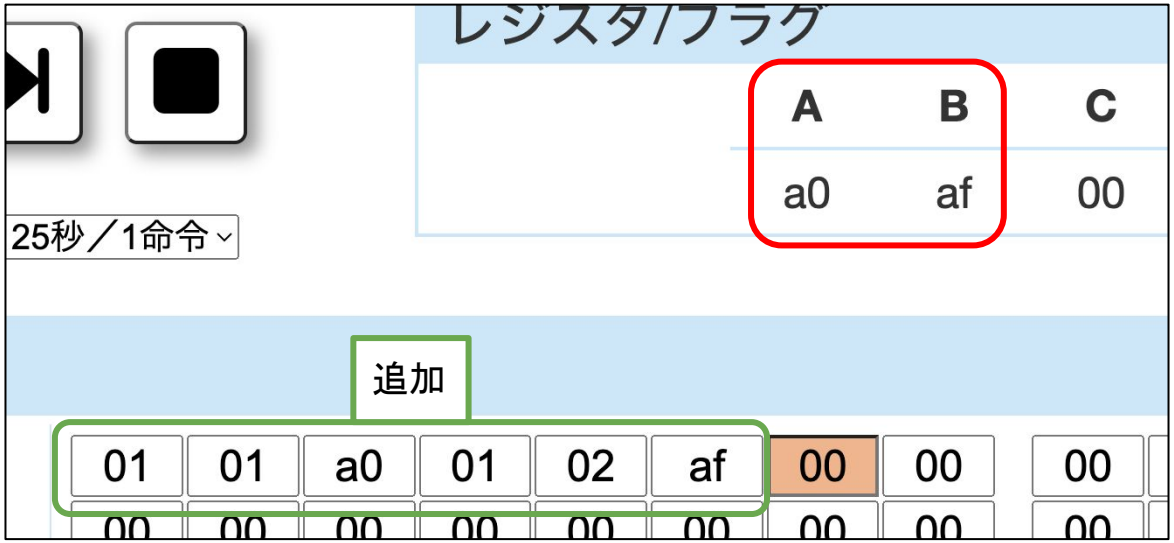
プログラム作成例

- ここではミクロン・シミュレータでプログラムを作成する手順の一例を紹介します
- プログラムの作成方法や手順は人によってバラバラなので、自分のやりやすい方法で作成してください
- 今回は、「0xA0～0xAF番地の各メモリに、その番地を格納するプログラム」を作成します



メモリマップの0xA0番地の行がこうなればOKです

1. レジスタAとBに、それぞれメモリマップの一番下の最初の番地と最後の番地を格納



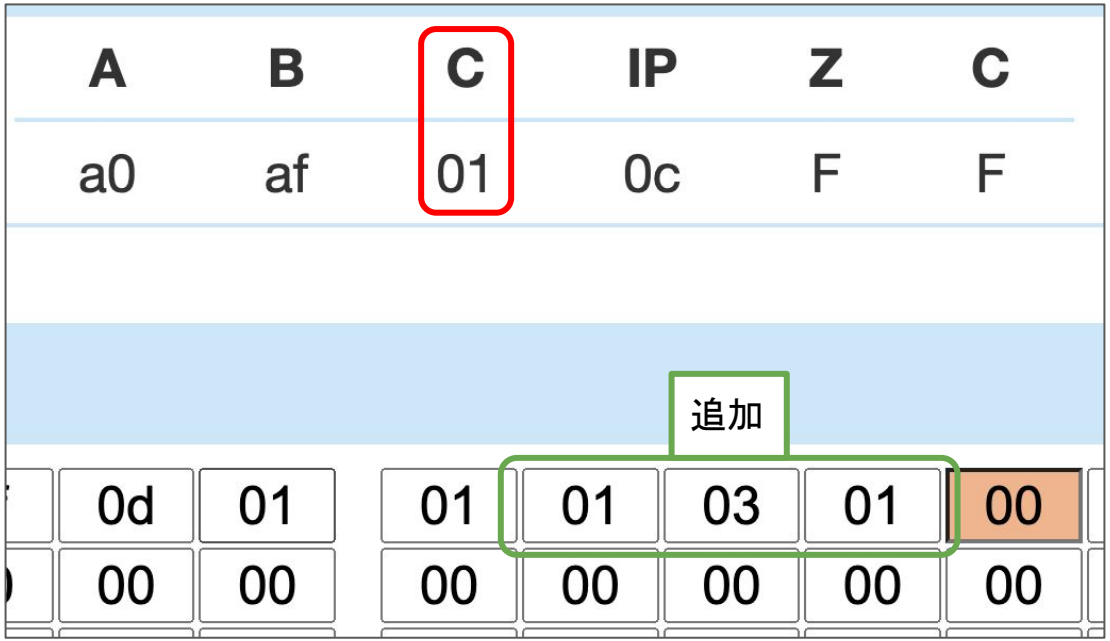
実行してみて、レジスタ／フラグの Aのところにa0、Bのところにafが表示されていたらOK

2.　レジスタAに格納した番地にレジスタAの値をそのままコピー

0x00番地	01	01	a0	01	02	af	0d	01	01	00	00	0
0x10番地	00	00	00	00	00	00	00	追加	00	00	00	0
0x20番地	00	00	00	00	00	00	00		00	00	00	0
0x30番地	00	00	00	00	00	00	00	00	00	00	00	0
0x40番地	00	00	00	00	00	00	00	00	00	00	00	0
0x50番地	00	00	00	00	00	00	00	00	00	00	00	0
0x60番地	00	00	00	00	00	00	00	00	00	00	00	0
0x70番地	00	00	00	00	00	00	00	00	00	00	00	0
0x80番地	00	00	00	00	00	00	00	00	00	00	00	0
0x90番地	00	00	00	00	00	00	00	00	00	00	00	0
0xA0番地	a0	00	00	00	00	00	00	00	00	00	00	0
0xB0番地	00	00	00	00	00	00	00	00	00	00	00	0

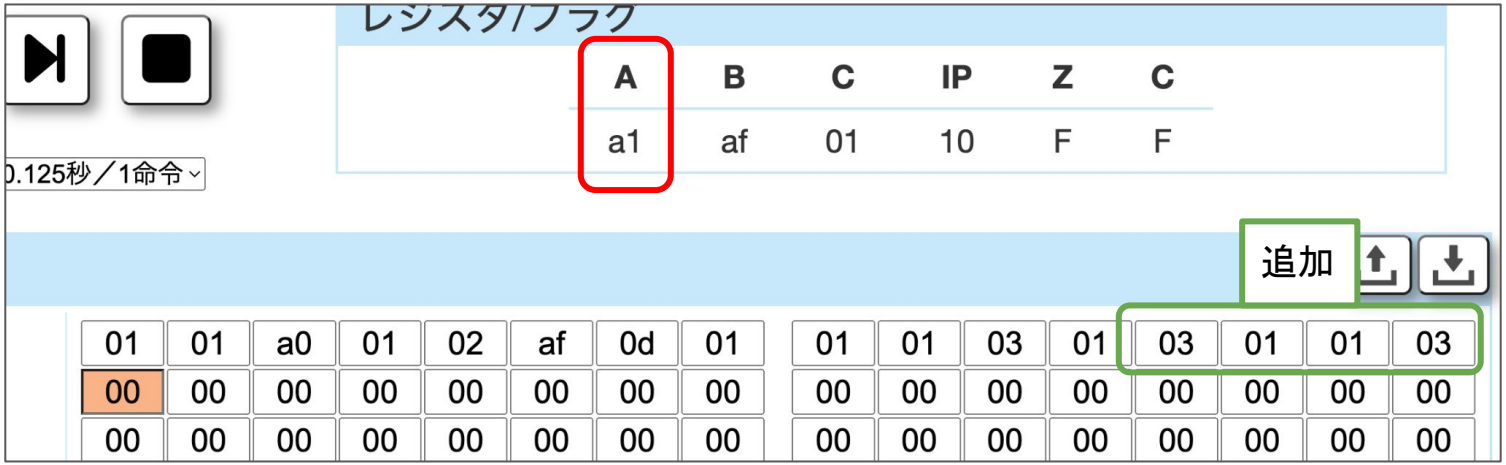
実行してみて、0xA0番地のメモリがa0になっていたらOK

3.　レジスタAの値を＋1するための値1をレジスタCに格納



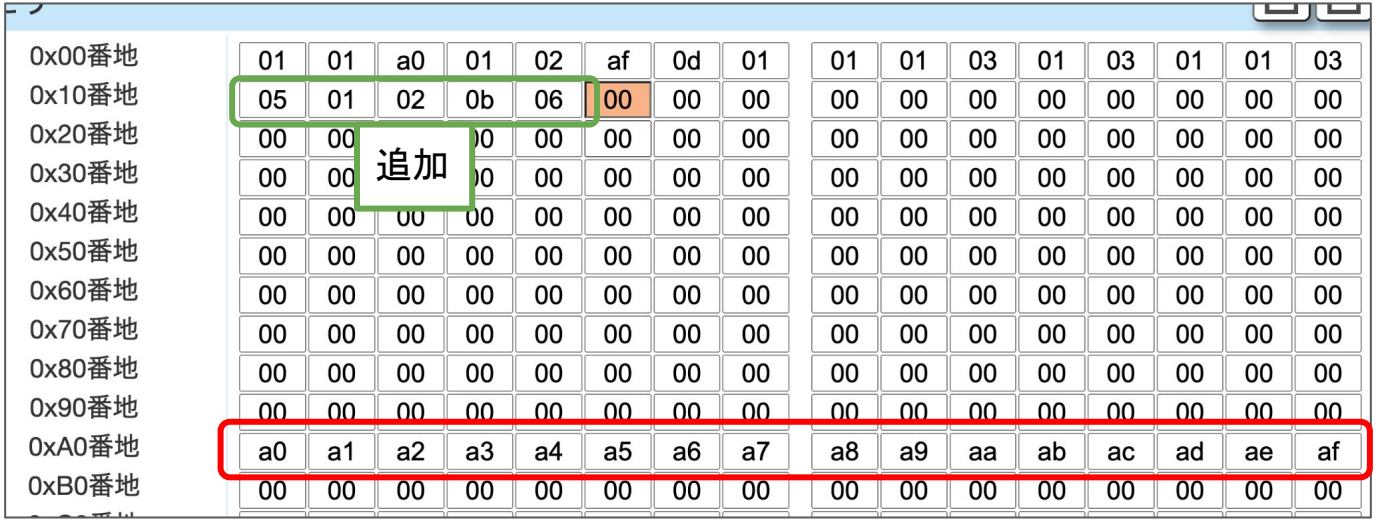
実行してみて、レジスタ／フラグの Cのところが01となっていたらOK

4.　レジスタAの値を＋1



実行してみて、レジスタ／フラグの Aのところがa0からa1に変わったらOK

5. レジスタAとBの値を比較し、A ≦ B が成り立つ時にstr命令が書かれた番地（0x06）にジャンプ



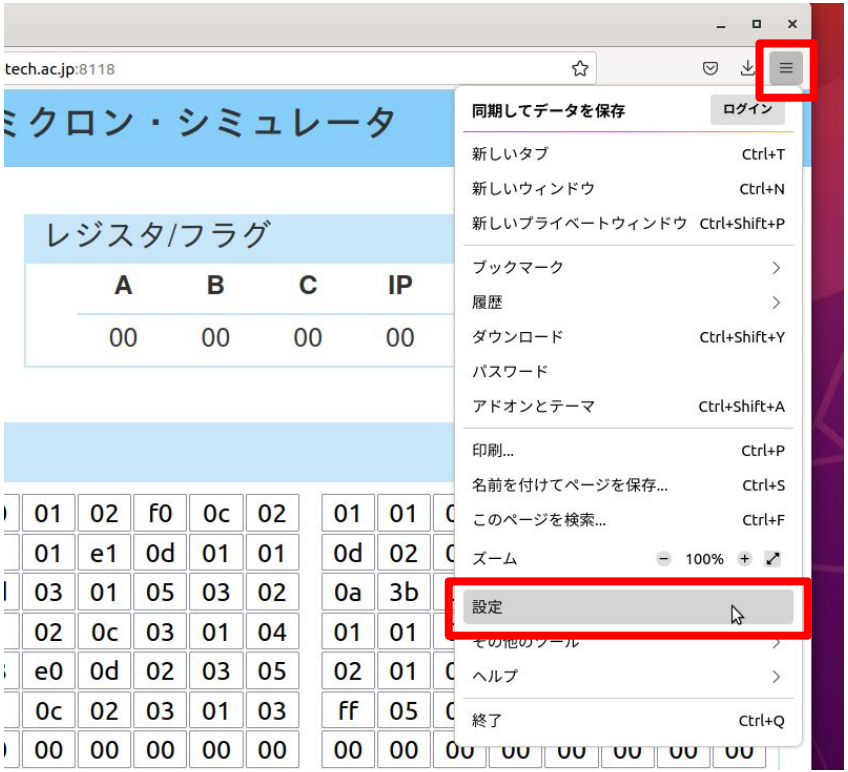
0x00番地	01	01	a0	01	02	af	0d	01	01	01	03	01	03	01	01	03
0x10番地	05	01	02	0b	06	00	00	00	00	00	00	00	00	00	00	00
0x20番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x30番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x50番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x60番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x70番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x80番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x90番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0xA0番地	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aa	ab	ac	ad	ae	af
0xB0番地	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

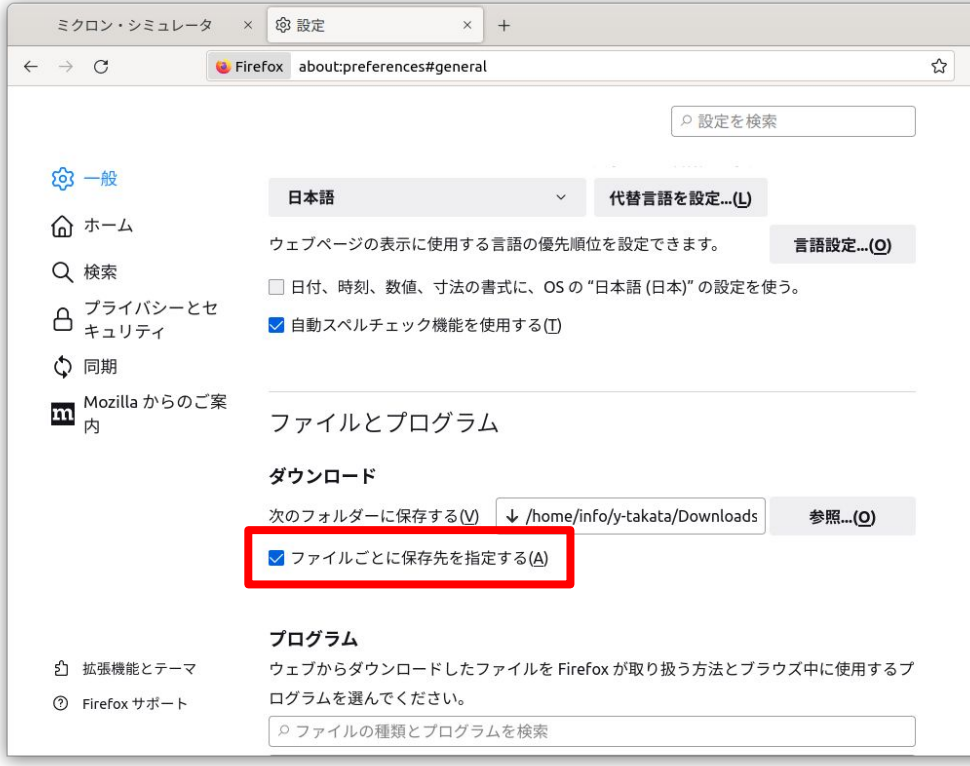
実行してみて、メモリマップの 0xA0番地の行が求めたい結果と一致したら OK

エラーについて

エラーメッセージ	原因
レジスタの番号が違います	● レジスタ番号を書くべきメモリに、間違った値を書いている
アドレスの指定に誤りがあります	● メモリのアドレスを書くべき場所に、存在しないメモリの番地を書いている
値に対応する命令がありません	● 命令に対応する値を書くべきメモリに、間違った値を書いている
メモリマップの上限を超えました	● プログラムがメモリマップの末尾を超えて続いている
16進数で値を設定してください	● 16進数では用いない文字を使用している
ファイルに16進数以外の値が含まれています	● アップロードしたファイル内に 16進数以外の文字が含まれている
コピーした内容がメモリを超えています	● メモリ編集で、ペーストによって書き込まれるメモリの末尾がメモリマップの末尾を超えている

Firefox: ファイルごとに保存先を指定する





ミクロン・シミュレータが動かなくなったとき

なんらかの操作を行って、突然ミクロン・シミュレータが固まって
動かなくなった時は、ミクロン・シミュレータ自体のバグです。

その場合はリロードしたら直りますが、書いていたプログラムは
全部消えるので、リロードする前にどこかに書き残しておいてください。

また、ミクロン・シミュレータに関するアンケートにバグについて書く欄があるので、
余裕があればどのような操作を行ったら動かなくなったかを
アンケートに書いていただけたら嬉しいです。