CSL
Computer Systems Laboratory

情報科学演習
プロセッサ演習
～第１週～

情報科学科
コンピュータシステム研究室

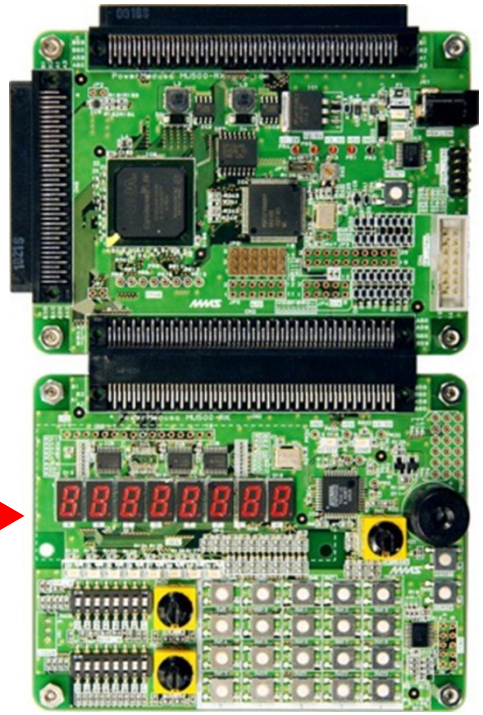
演習内容

- アセンブリ言語を用いたプログラム作成
  - MIPS準拠のシミュレータにて動作を確認
- FPGAボードに作成したプログラムを実装
  - シミュレータとの動作の違いを体験
  - プログラムの最適化を体験
- ソフトウェアとハードウェアの両方を学習

FPGA(Field Programmable Gate Array)

- 論理機能を書き換え可能なLSI
- ハードウェア記述言語を用いて回路設計可能
  - Verilog HDL
- PCから回路情報をダウンロード

回路情報



演習の流れ

第1週

■ ソートプログラムを作成

第2週

■ 命令メモリをVerilogファイルに変換

第3週

■ Quartus を用いてFPGAボードに情報転送

第4～5週

■ 作成したソートプログラムをボード上で動作

- ソートの実行時間を計測

アセンブリ言語

- 記号で書き表された言語(1命令:32bit=4byte)
- 機械語を人にわかりやすいように記述

機械語

```
1000110010100000
```

アセンブリ言語

```
add  A,B
```

- アセンブラ：アセンブリ言語を機械語に翻訳

アセンブリ言語

```
add  A,B
```

アセンブル

機械語

```
1000110010100000
```

- C言語のコンパイラはアセンブリ言語に翻訳

アセンブリ言語

- アセンブリ言語の記述の仕方

```
add   $t0, $t1, $t2        #$t0 = $t1+$t2
lw    $t0, 4($t3)          #$t0 = MEM($t3+4)
```

- $t1と$t2の値を加算した結果を$t0(レジスタ)に格納

アセンブリ言語

- アセンブリ言語の記述の仕方

```
add   $t0, $t1, $t2          #$t0 = $t1+$t2
lw    $t0, 4($t3)            #$t0 = MEM($t3+4)
```

- $t3の値に4を加算した番地の内容を$t0に格納

主なMIPS命令

- add：加算
- addi：即値加算
- lw：ロード
- sw：ストア
- slt：大小比較
- bne：等しくない場合、分岐
- beq：等しい場合、分岐
- j：ジャンプ
- nop：何もしない

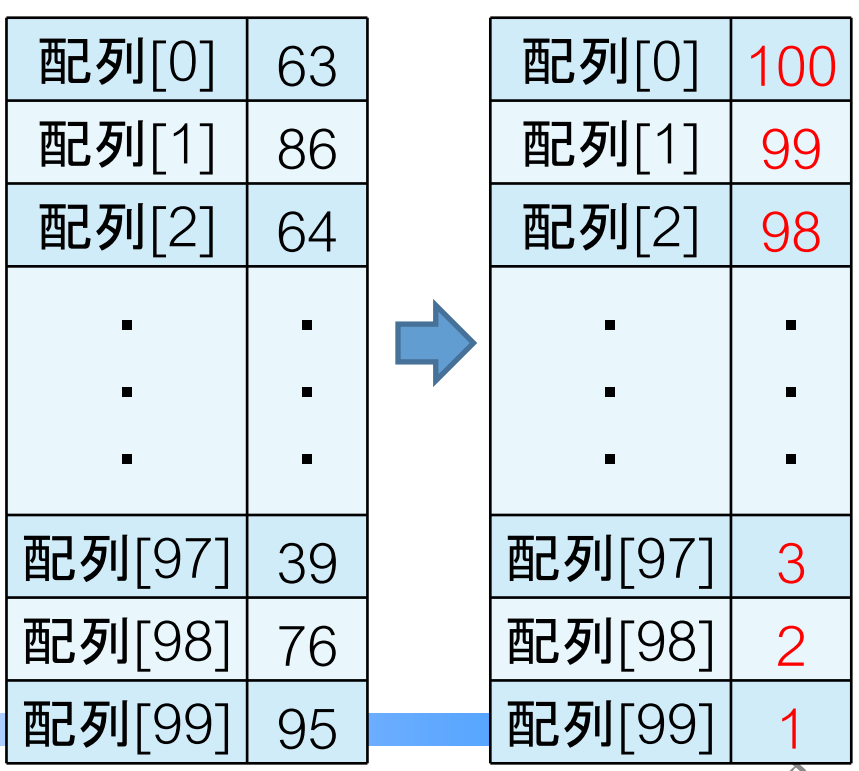
作成するアセンブリプログラム

■ 1~100の値がランダムに格納されている配列を降順に並び替えるプログラム

- ソートの種類は問わない

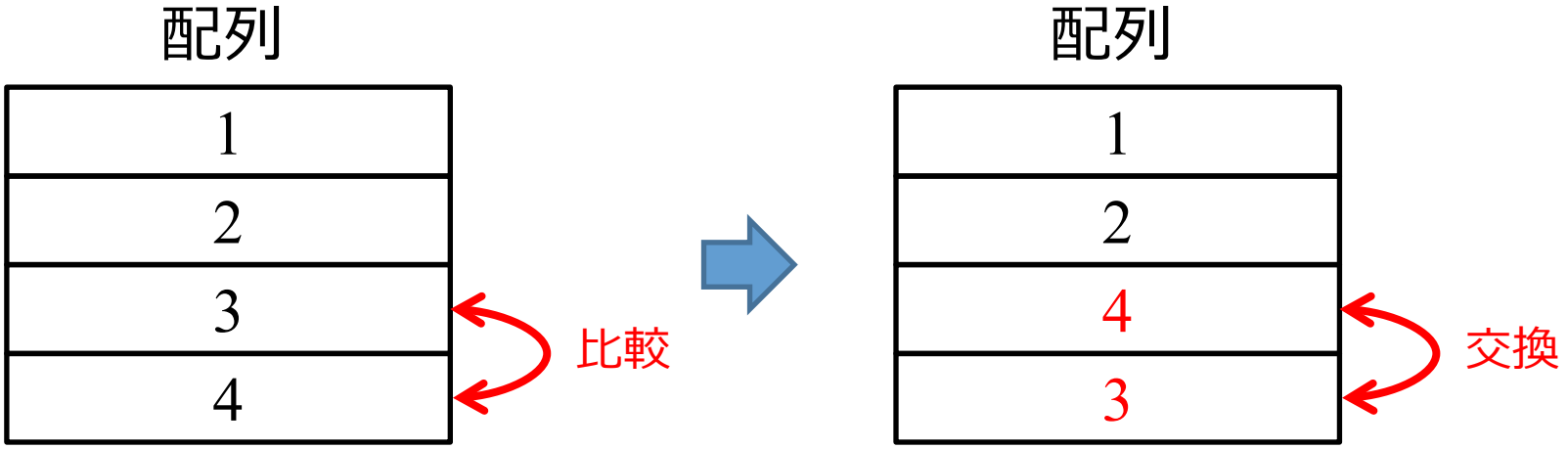
例：バブルソート ※**まずはこれを作成**
：挿入ソート
：クイックソート
：選択ソート
：オリジナルのソート

- 実行速度が速いプログラムを考えよう



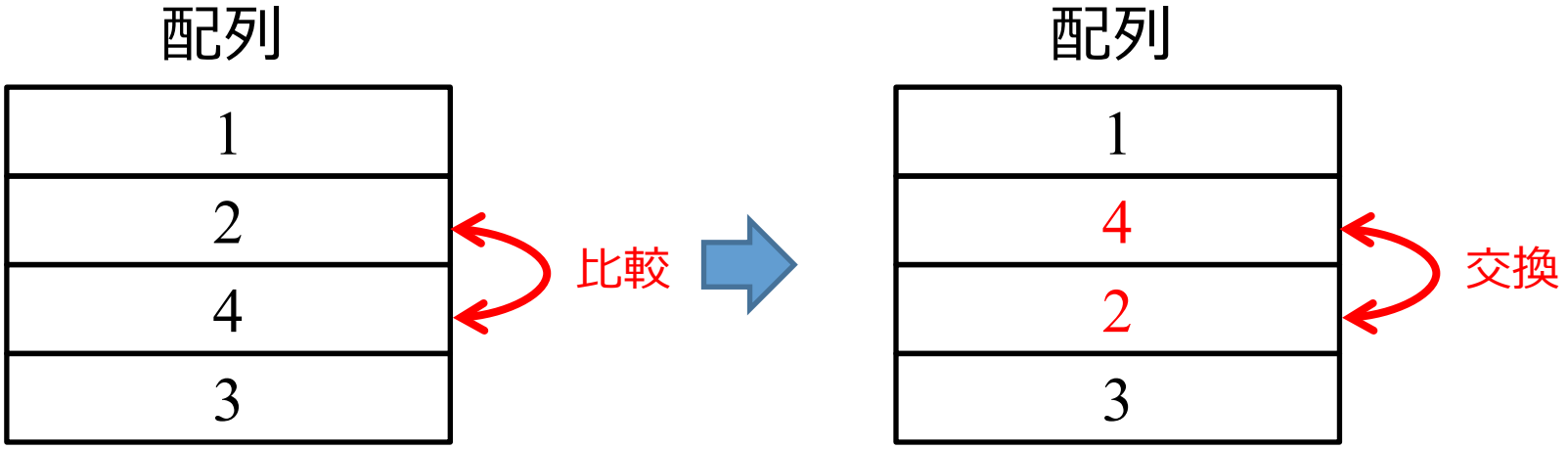
バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番上まで比較が終わると最大の値が一番上の要素に格納



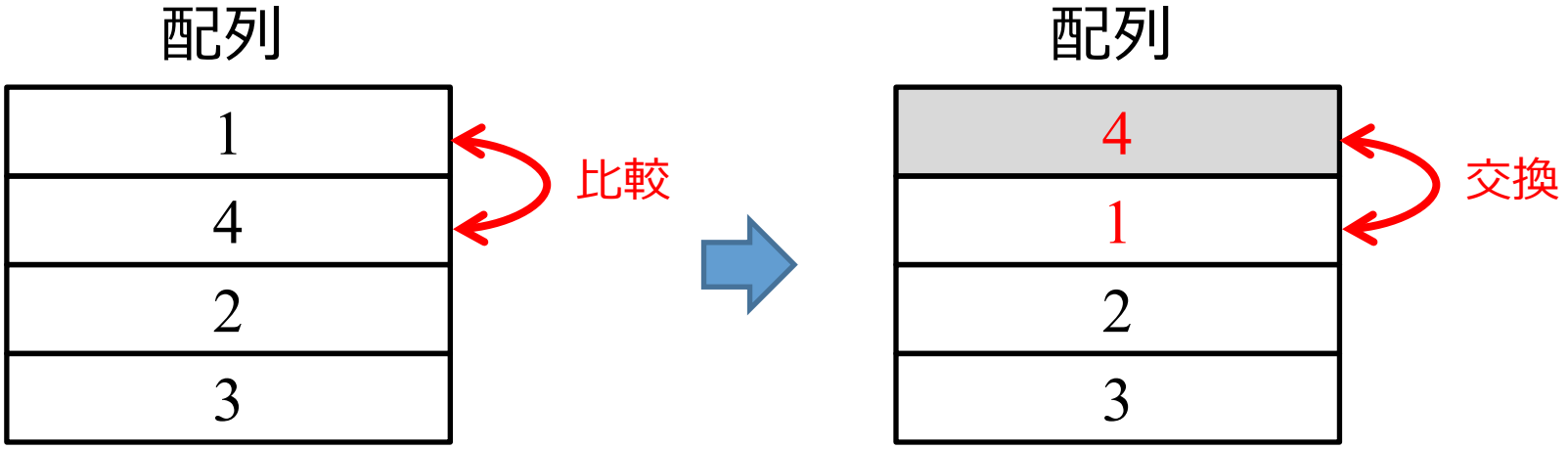
バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番上まで比較が終わると最大の値が一番上の要素に格納



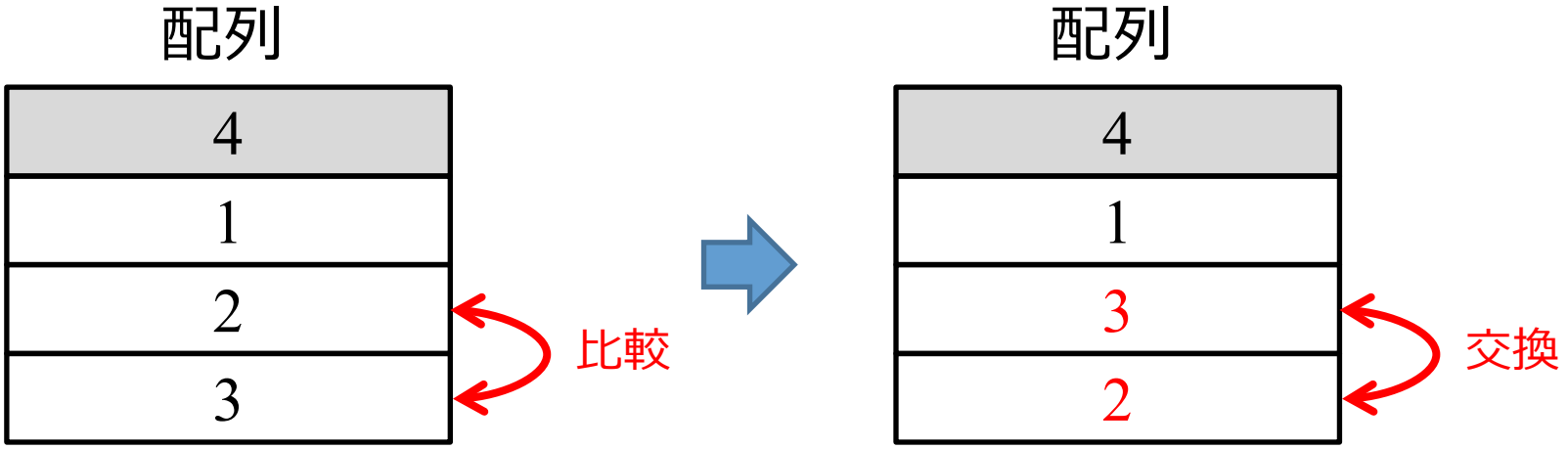
バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番上まで比較が終わると最大の値が一番上の要素に格納



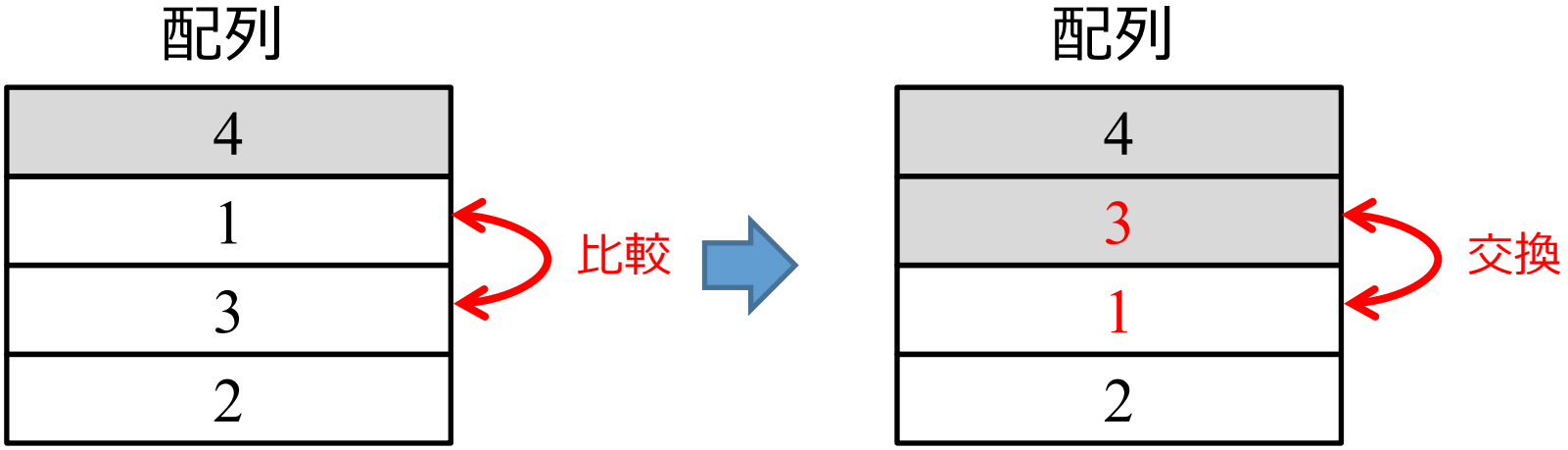
バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番下から比較を開始
- 並べ替えが完了した要素は比較しない



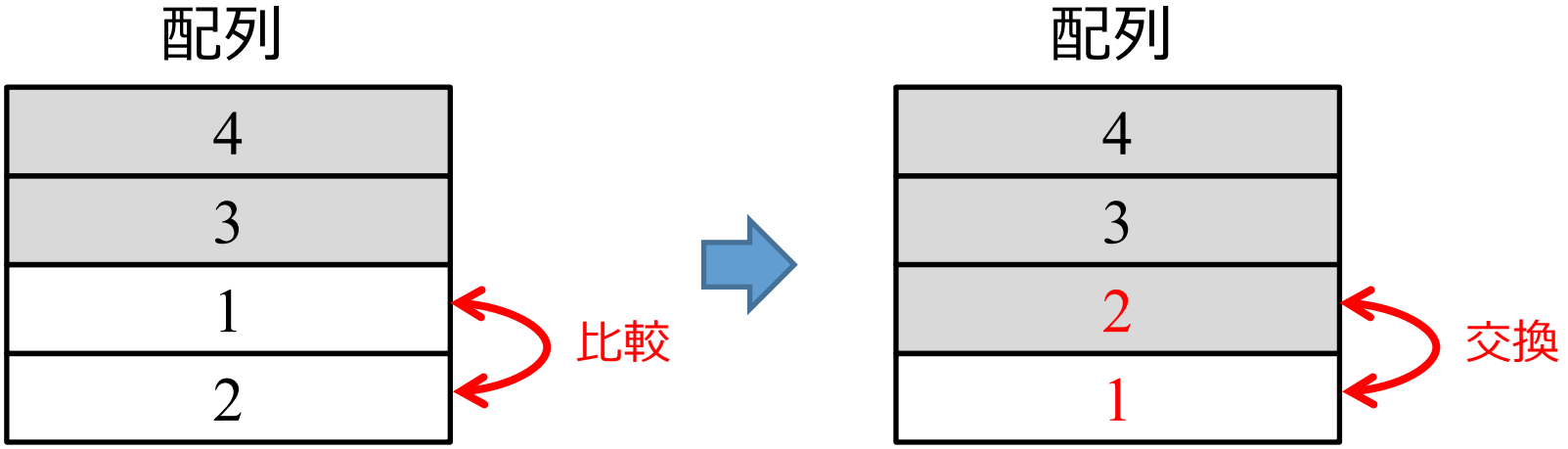
バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番下から比較を開始
- 並べ替えが完了した要素は比較しない



バブルソート

- 上の要素と比較し,上の要素の値が小さければ要素同士を交換
- 一番下から比較を開始
- 並べ替えが完了した要素は比較しない



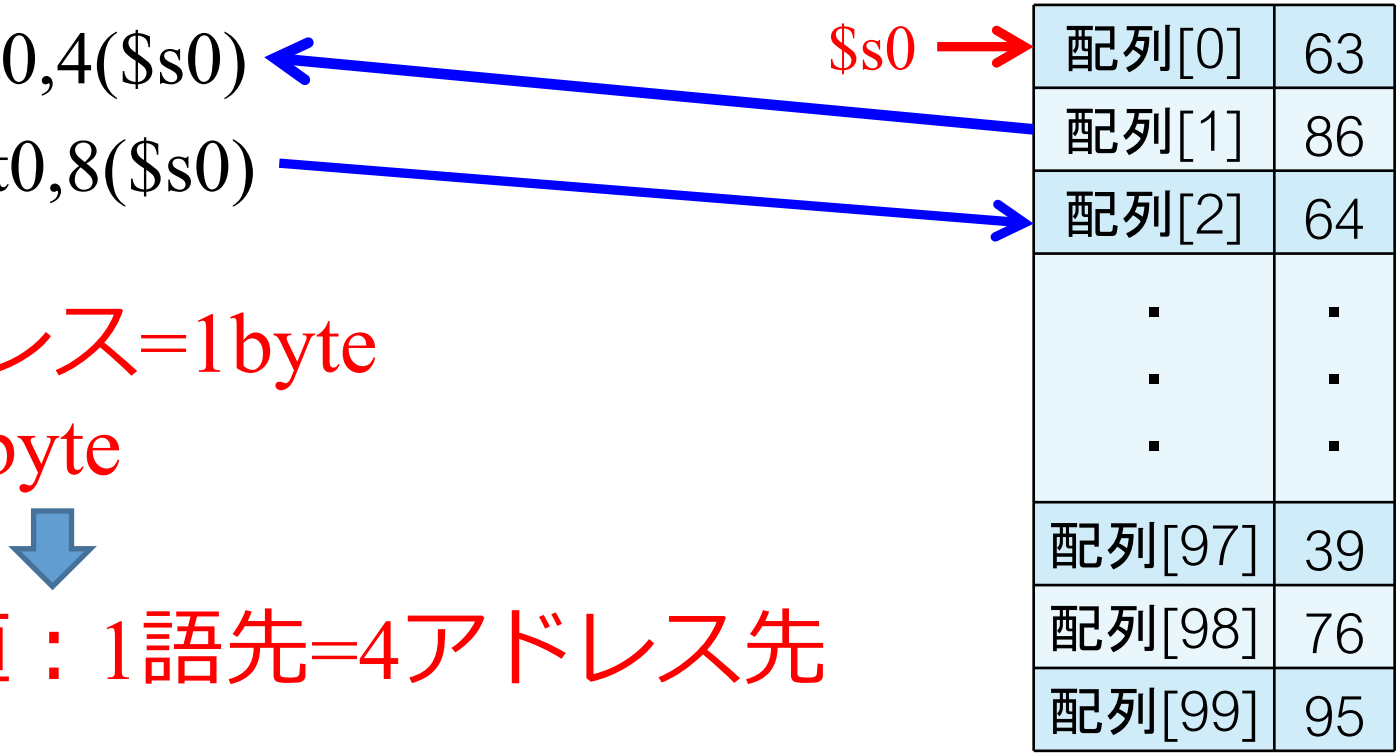
使用可能なレジスタ

- 回路で使用可能なレジスタの総数は32個
  このうち、主に以下を使用
  - $t0~$t7：汎用レジスタ(関数呼び出し時に消去)
  - $s1~$s7：汎用レジスタ(関数呼び出し時に退避)
  - $s0：データの先頭番地(更新しないように)
  - $zero：数値0が格納されているレジスタ

配列へのアクセスの方法

■ロード,ストア
(配列要素1にアクセスする場合)

- lw　　$t0,4($s0)
- sw　　$t0,8($s0)



※1アドレス=1byte
1語=4byte

次の値：1語先=4アドレス先

分岐命令,ジャンプ命令

- アドレスを直接記述する必要はない
- 分岐先にラベルを記述「ラベル名:」
- 命令のアドレス指定にはラベル名を記述

```
      beq  $x,$y,True
      addi $x, $x,1
      j    Exit
True: addi  $x, $x,-1
Exit: addi  $y, $y,1
```

```
if(x == y){
    x--;
} else {
    x++;
}
y++;
```

比較命令

- 大小比較の結果を1, 0 の値で格納
  - 第1入力と第2入力を入れ替えると結果が反転
- 分岐命令と組み合わせて条件分岐

x<yならz=1
x>=1ならz=0

z=1ならTrue
z=0なら次へ

```
slt    $z, $x, $y
bne  $z,$zero,True
addi $x, $x,1
j      Exit
True: addi  $x, $x,-1
Exit: addi  $y, $y,1
```

```
if(x < y){
    x--;
} else {
    x++;
}
y++;
```

ソートプログラム作成のポイント

- 同じ種類のソートでもプログラムによって実行速度は異なる
  - 命令数により実行速度が変化
- コンパイラが行っている最適化を考慮

```
x = 10;
y = 9;
if( x  == y )
{
    x = x + 1;
}
else
{
    y = y + 1;
}
```

```
        addi  $t0, $zero, 10
        addi  $t1, $zero, 9
        beq   $t0, $t1, TRUE
        j     ELSE
TRUE:   addi  $t0. $t0, 1
        j     EXIT
ELSE:   addi  $t1, $t1, 1
EXIT:
```

条件を逆に

1命令削減

```
        addi  $t0, $zero, 10
        addi  $t1, $zero, 9
        bne   $t0, $t1, ELSE
        addi  $t0. $t0, 1
        j     EXIT
ELSE:   addi  $t1, $t1, 1
EXIT:
```

MARSについて

■MIPSアセンブリ言語でプログラミングするための対話型開発環境

- エディタ機能
- シミュレーション機能

■ダウンロード

- http://courses.missouristate.edu/KenVollmar/MARS/

MARSの使い方1

■MARSの起動方法

- 端末上で「java –jar Mars4_5.jar」を入力

■Javaがインストールされていない場合

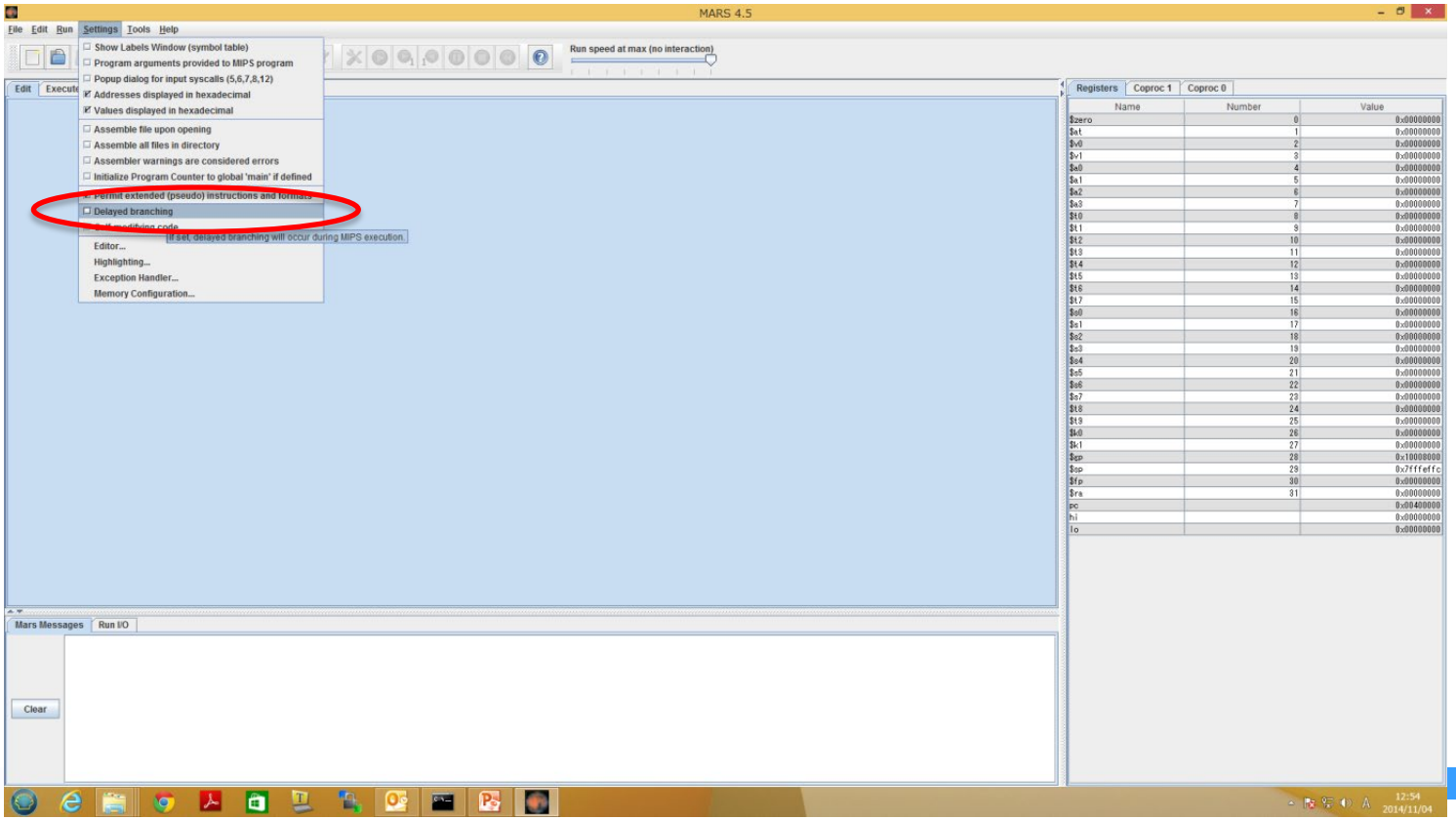
- Ubuntu:「sudo apt install default-jdk」を入力
- Windows: https://www.microsoft.com/openjdk
  - 上記URLからダウンロードしてインストール
- その後にMARSを起動

MARSの使い方2

※遅延分岐は次週以降に使用

■MARSの設定(遅延分岐を設定しない場合※)

- Settings > Delayed branchingのチェックは外す



MARSの使い方3

■アセンブリプログラム作成方法

- File > New
- プログラムの最後に以下の記述を行う

```
…
li      $v0,  10
syscall
```

return 0;

MARSの使い方4

■ファイルの保存方法

- File > Save
- ファイル名入力 > 保存(拡張子:.asm)

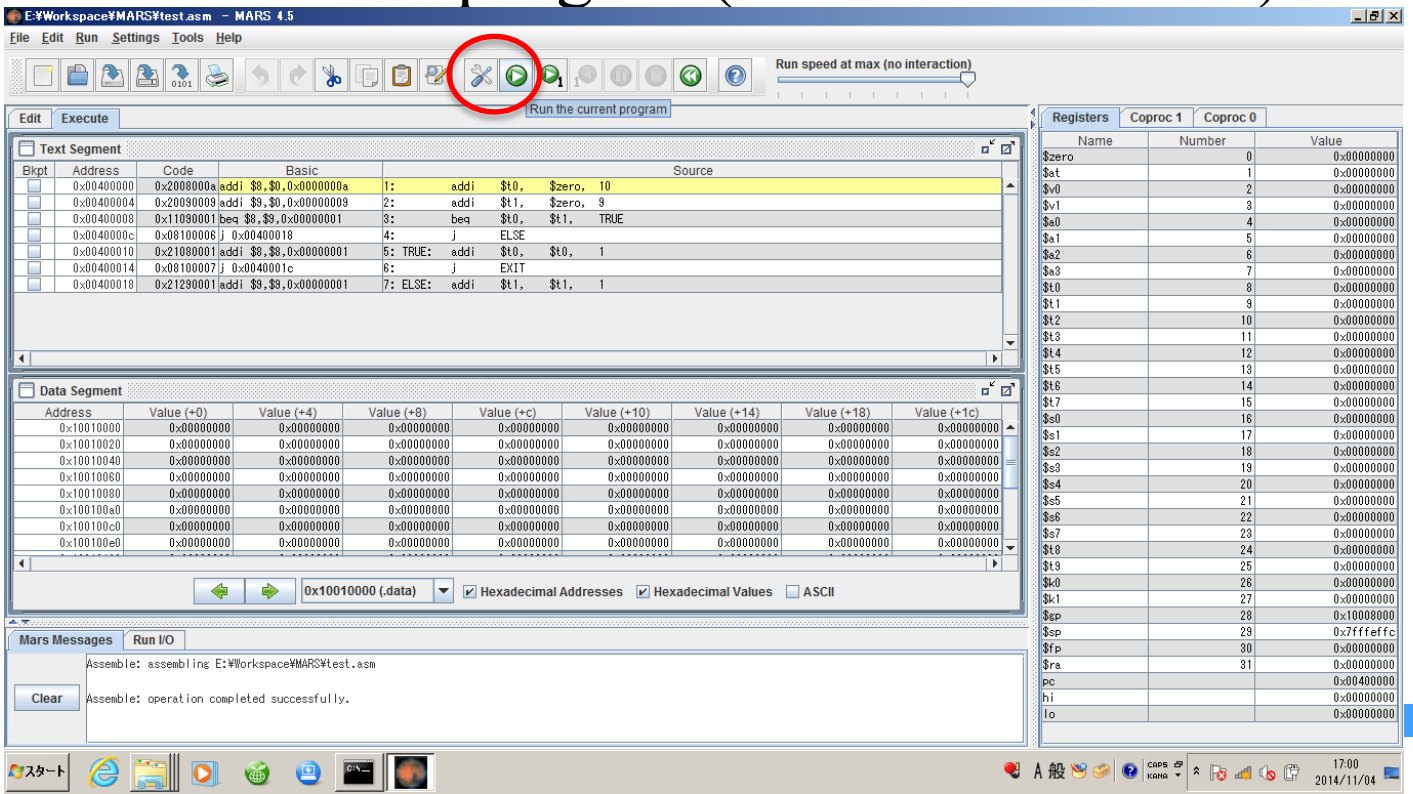
■ファイルの読み込み方法

- File > Open > ファイル選択 > 開く(拡張子:.asm)

MARSの使い方5

■実行方法

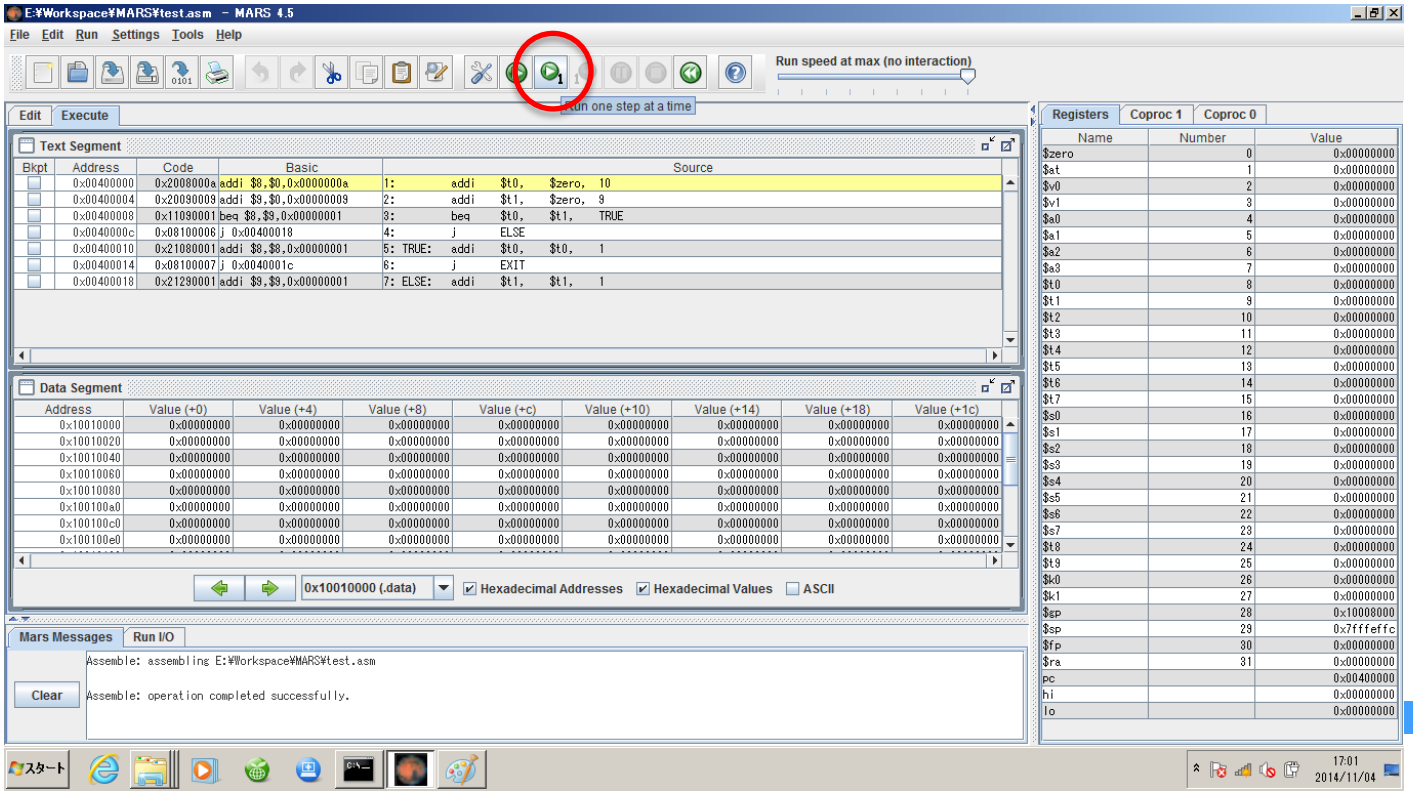
- Run > Assemble (機械語に変換)
- Run the current program(プログラム実行)



MARSの使い方6

■ステップ実行

- Run one step at a time(ステップごとに命令実行)



第１週の課題

- 分岐命令の動作確認
  - branch1.asm: 条件に応じた分岐の動作確認
- ロード/ストア命令の動作確認
  - ldst1.asm: メモリのロード/ストアの動作確認
  - swap1.asm: データのスワップ処理の動作確認
- バブルソートのプログラム作成
  - bubble004.asm: 少数(4件)のデータでの動作確認
  - bubble100.asm: 100件のデータでの動作確認

■ branch1.asm

以下のプログラムの実行結果、および、1, 2行目の値や、3行目の beq を bne に変更した場合の結果も確認せよ。

```
        addi    $t0,    $zero,  10      # X
        addi    $t1,    $zero,  9       # Y
        beq     $t0,    $t1,    TRUE    # 等しければTRUEへ

        j       ELSE                    # ELSEへ

TRUE:
        addi    $t0,    $t0,    1       # X++
        j       EXIT                    # EXITへ

ELSE:
        addi    $t1,    $t1,    1       # Y++
EXIT:
        li      $v0,    10              # プログラムの終了
        syscall
```

■ ldst1.asm

以下のプログラムの実行結果、および、8, 9行目のロード先のレジスタをそれぞれ$t1, $t0と入れ替えた場合の結果も確認せよ。

```
        lui     $s0,    0x00001001      # データの先頭番地A
        addi    $t2,    $zero,  10      # X
        addi    $t3,    $zero,  9       # Y
        sw      $t2,    0($s0)          # Xの値を(A+0)番地にストア
        sw      $t3,    4($s0)          # Yの値を(A+4)番地にストア
        lw      $t0,    0($s0)          # (A+0)番地の値をロード
        lw      $t1,    4($s0)          # (A+4)番地の値をロード
        sw      $t0,    8($s0)          # (A+8)番地にストア
        sw      $t1,    12($s0)         # (A+12)番地にストア

        li      $v0,    10              # プログラムの終了
        syscall
```

■ swap1.asm

以下のプログラムを実行して、
最大値のデータがどのように移動するかを確認せよ。

```
        ...
        add     $s1,    $zero,  $s0     # データの先頭番地A
        addi    $s2,    $s1,    8       # k = A+8
LOOP:
        slt     $t0,    $s2,    $s0     # k < A なら 1
        bne     $t0,    $zero,  EXIT    # k < A なら EXITへ
        lw      $t0,    0($s2)          # $t0に(k+0)番地の値をロード
        lw      $t1,    4($s2)          # $t1に(k+4)番地の値をロード
        sw      $t0,    4($s2)          # $t0の値を(k+4)番地にストア
        sw      $t1,    0($s2)          # $t1の値を(k+0)番地にストア
        addi    $s2,    $s2,    -4      # 1つ前のデータへ
        j       LOOP                    # LOOPへ
EXIT:
        li      $v0,    10              # プログラムの終了
        syscall
```

■ bubble004.asm, bubble100.asm
バブルソートのプログラムを完成させて、
実行結果を確認せよ。以下の部分に記述すること。

```
        ...
Sort0:
        ########################################
        #↓以下に並び替えのプログラム記述
        #     ここだけに追加記入すること
        #↑ここまで
        ########################################
        ...
```

■ ???sort.asm

バブルソート以外のプログラムを完成させて、
実行結果を確認せよ。以下の部分に記述すること。

```
        ...
Sort0:
        ########################################
        #↓以下に並び替えのプログラム記述
        #     ここだけに追加記入すること
        #↑ここまで
        ########################################
        ...
```

例：バブルソート

：挿入ソート

：クイックソート

：選択ソート

：・・・

算術演算命令、ロード／ストア命令

命令	ニモニック	意味
add	`add $s1,$s2,$s3`	`$s1 ← $s2+$s3`
add unsigned	`addu $s1,$s2,$s3`	`$s1 ← $s2+$s3`
add immediate	`addi $s1,$s2,4`	`$s1 ← $s2 + 4`
add immediate unsigned	`addiu $s1,$s2,4`	`$s1 ← $s2 + 4`
subtract	`sub $s1,$s2,$s3`	`$s1 ← $s2-$s3`
subtract unsigned	`subu $s1,$s2,$s3`	`$s1 ← $s2-$s3`

命令	ニモニック	意味
load word	`lw $s1,20($s2)`	`$s1 ← MEM[$s2+20]`
store word	`sw $s1,20($s2)`	`MEM[$s2+20] ← $s1`
load upper immediate (上位ビットのロード)	`lui $s1,0xABCD`	`$s1 ← 0xABCD0000`

論理演算命令

命令	ニモニック	意味
and	`and $s1,$s2,$s3`	`$s1 ← $s2 & $s3`
or	`or $s1,$s2,$s3`	`$s1 ← $s2 | $s3`
xor	`xor $s1,$s2,$s3`	`$s1 ← $s2 ^ $s3`
nor	`nor $s1,$s2,$s3`	`$s1 ← ~($s2 | $s3)`
and immediate	`andi $s1,$s2,20`	`$s1 ← $s2 & 20`
or immediate	`ori $s1,$s2,20`	`$s1 ← $s2 | 20`
xor immediate	`xori $s1,$s2,20`	`$s1 ← $s2 ^ 20`

命令	ニモニック	意味
shift left logical	`sll $s1,$s2,10`	`$s1 ← $s2 << 10`
shift right logical (符号なし)	`srl $s1,$s2,10`	`$s1 ← $s2 >> 10`
shift right arithmetic (符号付き)	`sra $s1,$s2,10`	`$s1 ← $s2 >> 10`

比較命令

命令	ニモニック	意味
set on less than (符号付き)	`slt  $s1,$s2,$s3`	`if($s2 < $s3) $s1 = 1;` `else $s1 = 0;`
set on less than unsigned (符号なし)	`sltu $s1,$s2,$s3`	`if($s2 < $s3) $s1 = 1;` `else $s1 = 0;`
set on less than immediate (符号付き)	`slti  $s1,$s2,20`	`if($s2 < 20) $s1 = 1;` `else $s1 = 0;`
set on less than immediate unsigned (符号なし)	`sltiu $s1,$s2,20`	`if($s2 < 20) $s1 = 1;` `else $s1 = 0;`

条件分岐命令、ジャンプ命令
コール命令、リターン命令

命令	ニモニック	意味
branch on equal	`beq $s1,$s2,25`	`if($s1==$s2)` `PC ← PC+4+100`
branch on not eq.	`bne $s1,$s2,25`	`if($s1!=$s2)` `PC ← PC+4+100`
jump	`j 2500`	`PC ← 10000`

命令	ニモニック	意味
jump and link	`jal 2500`	`$ra ← PC+4` `PC ← 10000`
jump register	`jr $ra`	`PC ← $ra`