アセンブリ言語プログラミング(1)

第1週目

担当　情報システム部門　徳光政弘
2025年4月11日

今日の内容

- 1週目
  - MIPSアーキテクチャの概要
  - 命令セットの考え方
  - プログラミング環境の構築
  - 簡単なプログラミング

今日の内容

- 2週目
  - 繰り返し処理
  - 配列
  - サブルーチン

実習内容

- 3週目
  - 関数の実装
  - 応用プログラミング

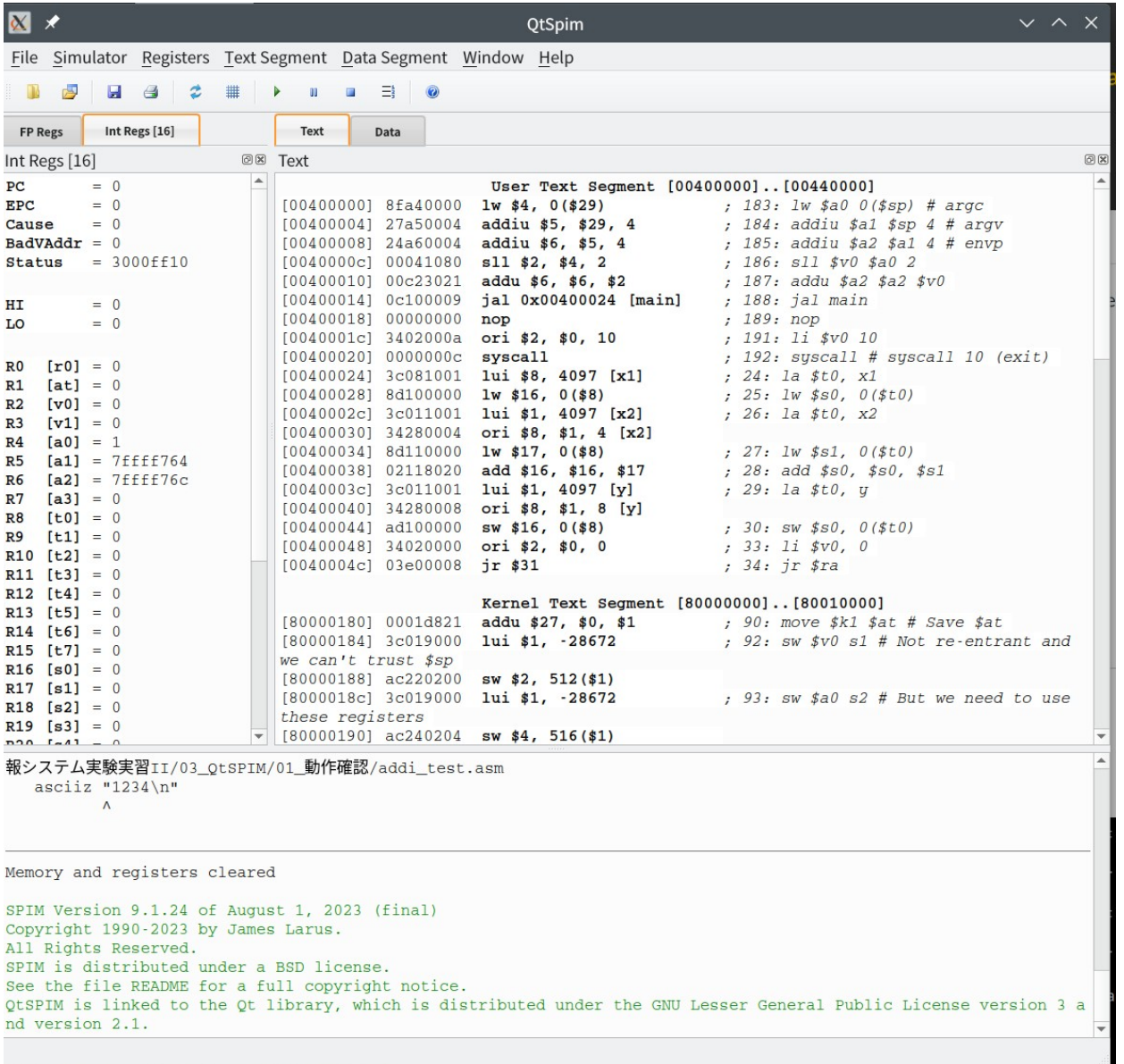
今日の内容

- 1週目
  - プログラミング環境の構築
  - MIPSアーキテクチャの概要
  - 命令セットの考え方
  - 簡単なプログラミング

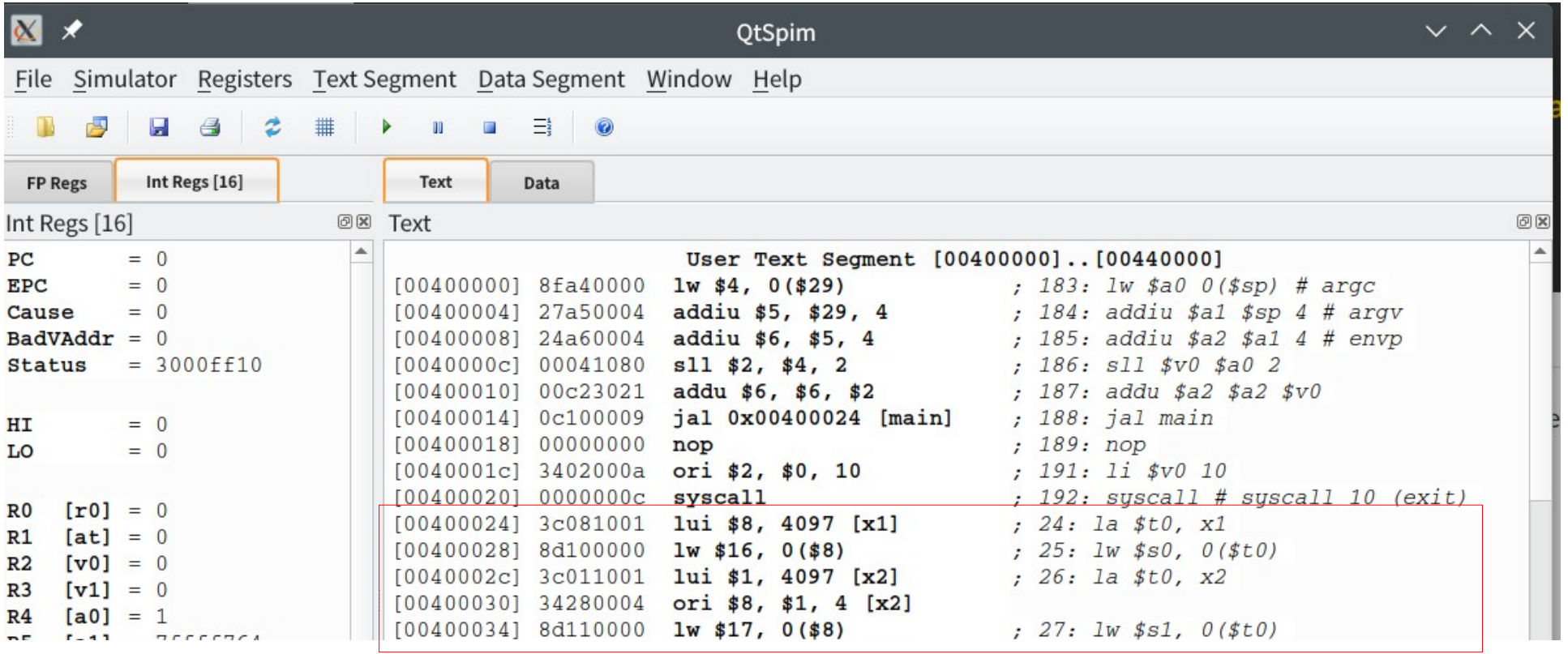
QtSPIM

- MIPSの命令を仮想環境で実行できるエミュレータ
- いろいろなシミュレータがあるうちのひとつ
- オープンソースで開発されている
- 自由に利用できる
- Windows、MacOS、Linuxで実行できる

QtSPIM

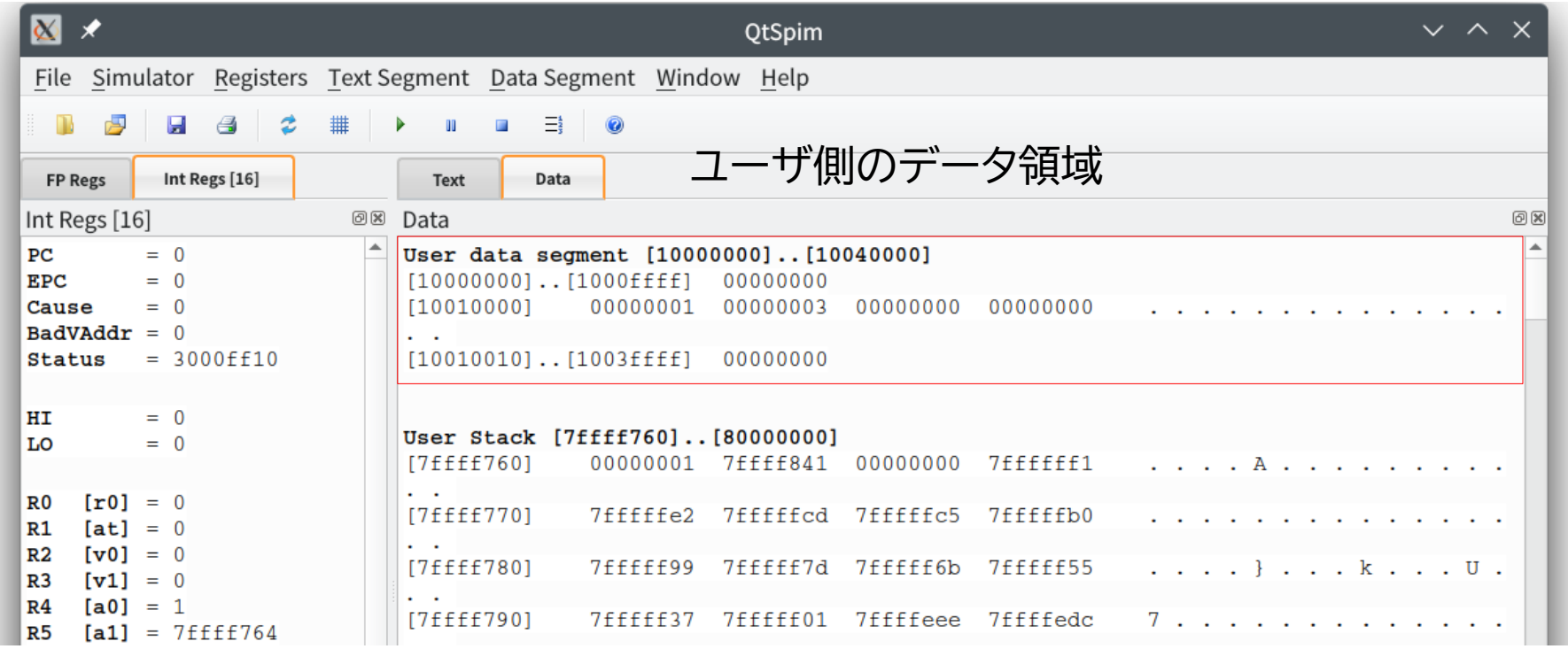

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Text
Data
Int Regs [16]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffff764
R6 [a2] = 7ffff76c
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
Text
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 3c081001 lui $8, 4097 [x1] ; 24: la $t0, x1
[00400028] 8d100000 lw $16, 0($8) ; 25: lw $s0, 0($t0)
[0040002c] 3c011001 lui $1, 4097 [x2] ; 26: la $t0, x2
[00400030] 34280004 ori $8, $1, 4 [x2]
[00400034] 8d110000 lw $17, 0($8) ; 27: lw $s1, 0($t0)
[00400038] 02118020 add $16, $16, $17 ; 28: add $s0, $s0, $s1
[0040003c] 3c011001 lui $1, 4097 [y] ; 29: la $t0, y
[00400040] 34280008 ori $8, $1, 8 [y]
[00400044] ad100000 sw $16, 0($8) ; 30: sw $s0, 0($t0)
[00400048] 34020000 ori $2, $0, 0 ; 33: li $v0, 0
[0040004c] 03e00008 jr $31 ; 34: jr $ra
Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1)
報システム実験実習II/03_QtSPIM/01_動作確認/addi_test.asm
asciiz "1234\n"
^
Memory and registers cleared
SPIM Version 9.1.24 of August 1, 2023 (final)
Copyright 1990-2023 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

QtSPIM

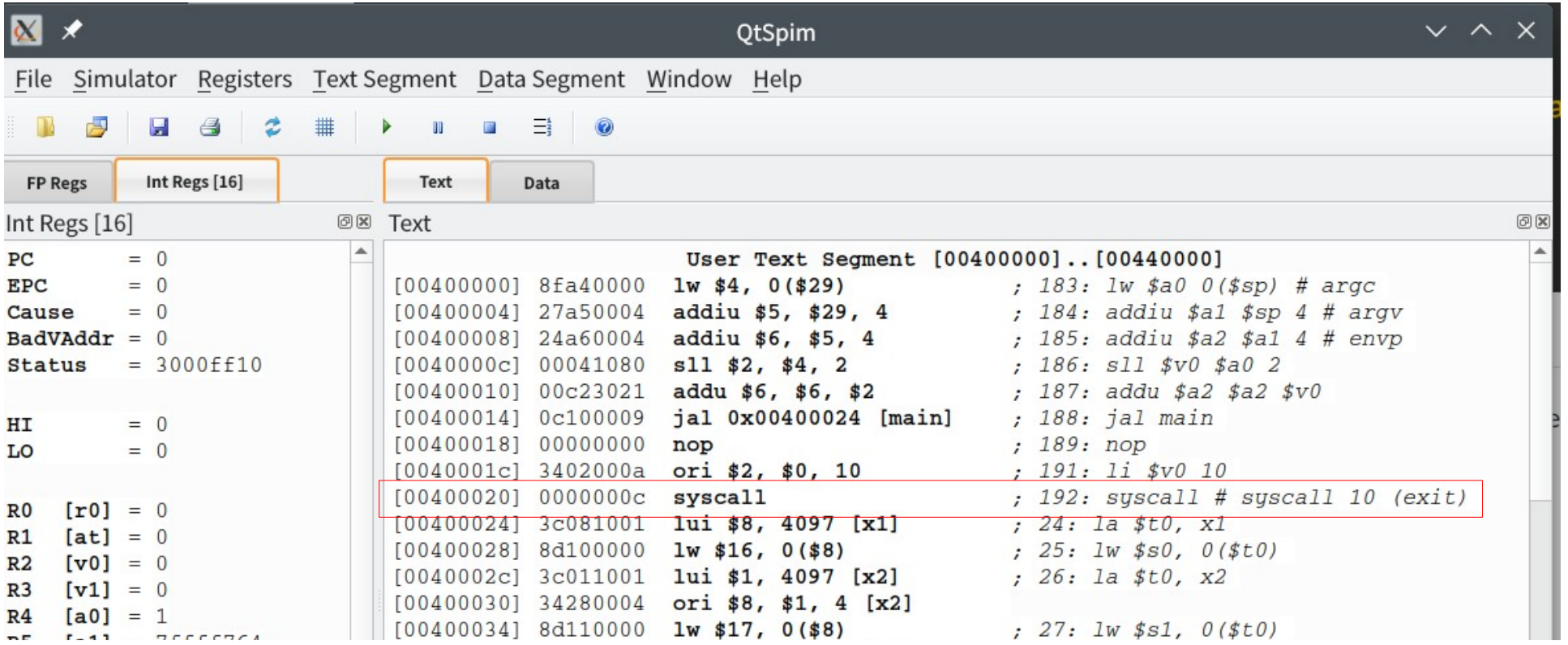


ユーザ側のプログラムの範囲

QtSPIM

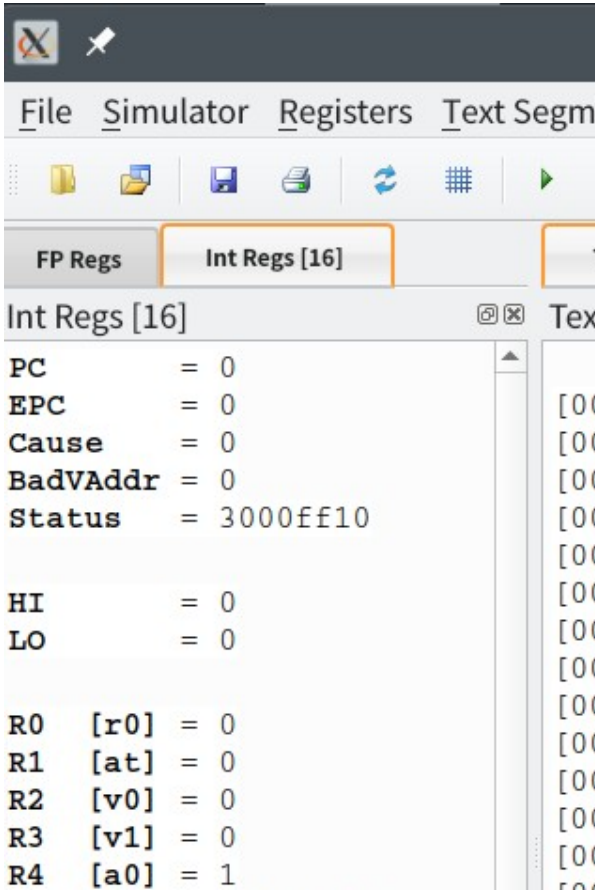

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Text
Data
ユーザ側のデータ領域
Int Regs [16]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffff764
Data
User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 00000001 00000003 00000000 00000000
[10010010]..[1003ffff] 00000000
User Stack [7ffff760]..[80000000]
[7ffff760] 00000001 7fff841 00000000 7ffffff1
[7ffff770] 7ffffe2 7fffffcd 7fffffc5 7fffffb0
[7ffff780] 7fffff99 7fffff7d 7fffff6b 7fffff55
[7ffff790] 7fffff37 7fffff01 7ffffeee 7ffffedc

QtSPIM

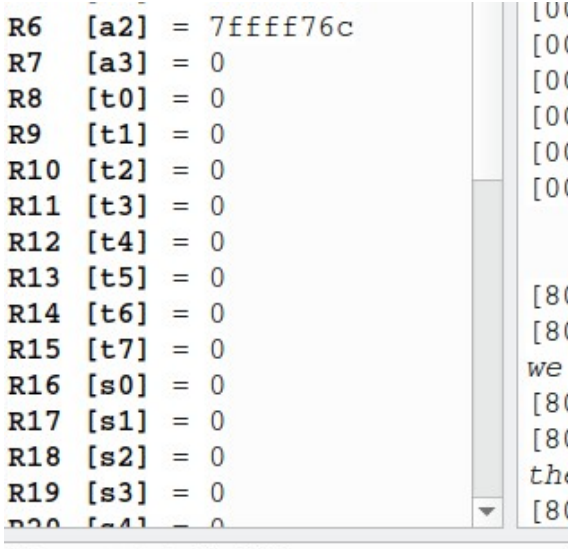


「syscall」以下から実行される

QtSPIM

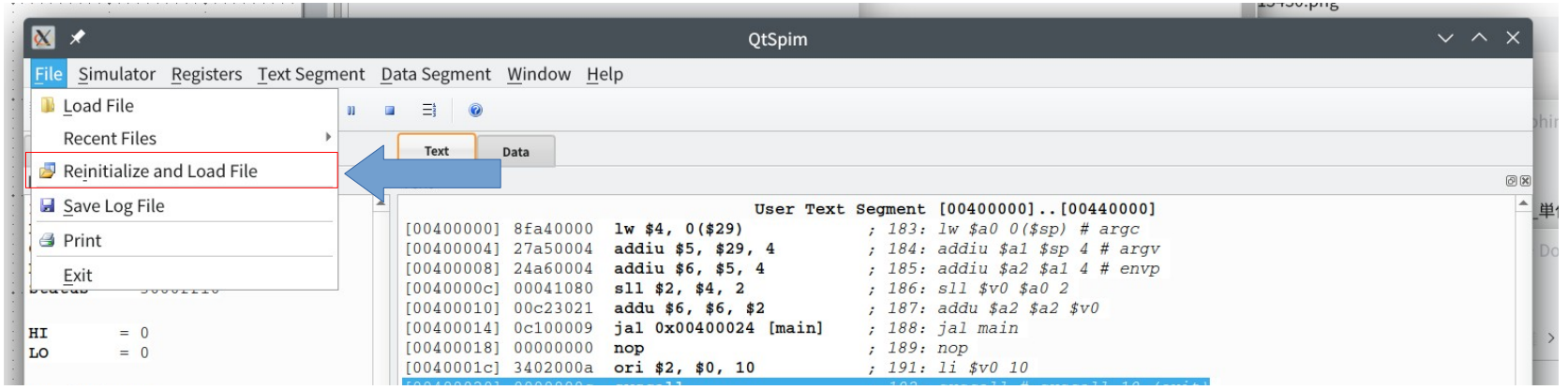


画面左側はレジスタの値が表示される

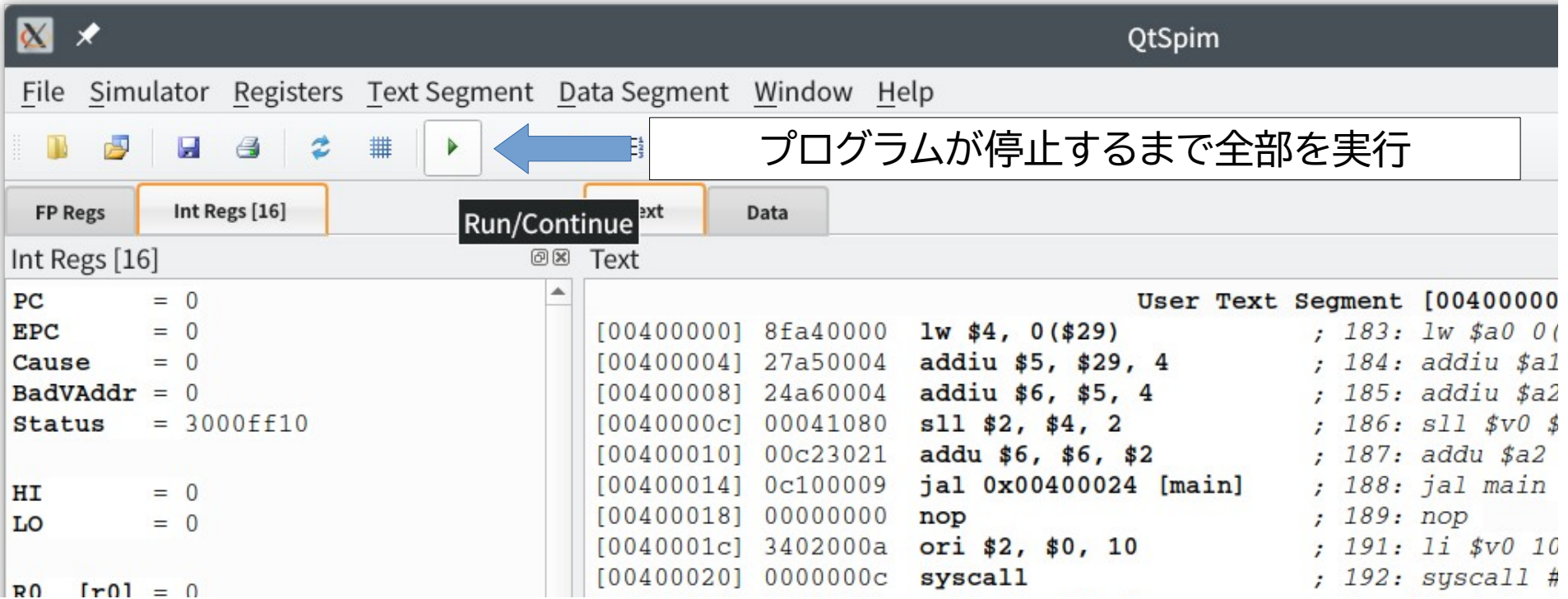


アセンブリプログラムの読み込み

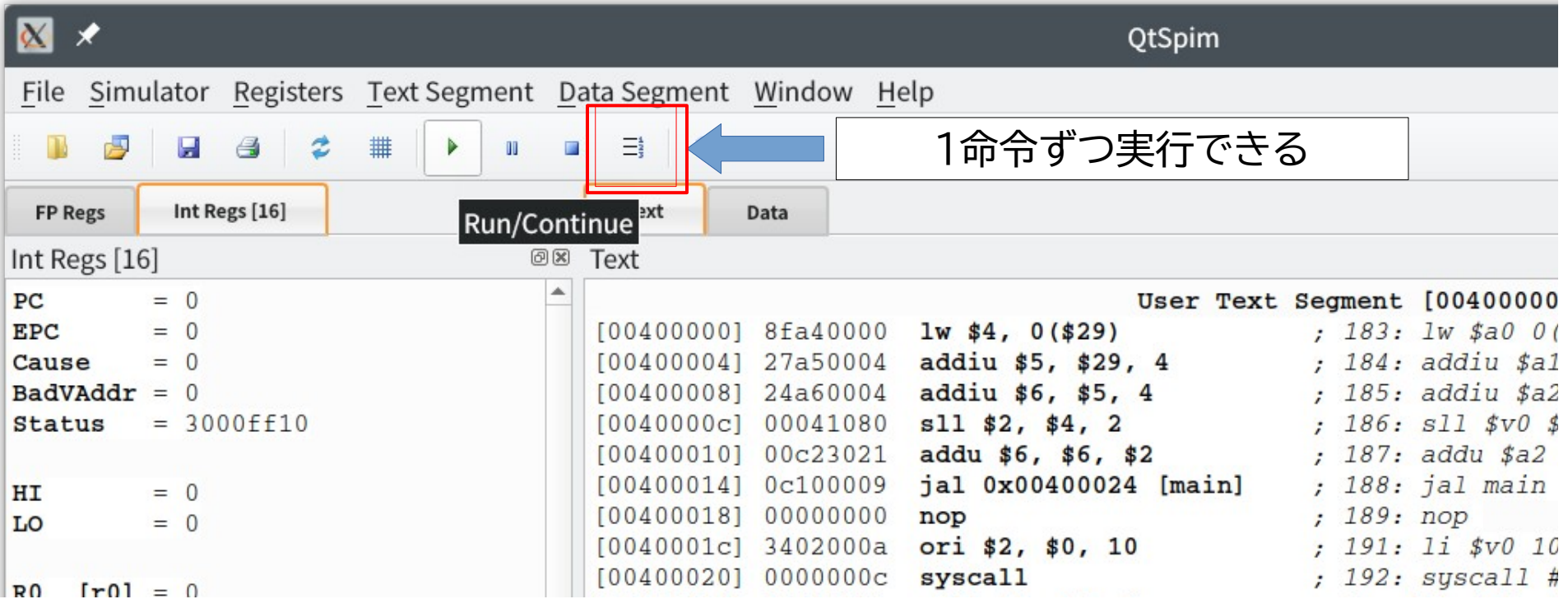
- 「File」->「Reinitialize and Load File」



アセンブリプログラムの実行



アセンブリプログラムの実行



アセンブリプログラムの例

```
        .globl  main
main:
        addi $t0, $zero, 1
        addi $t0, $t0, 1
        jr $ra
```

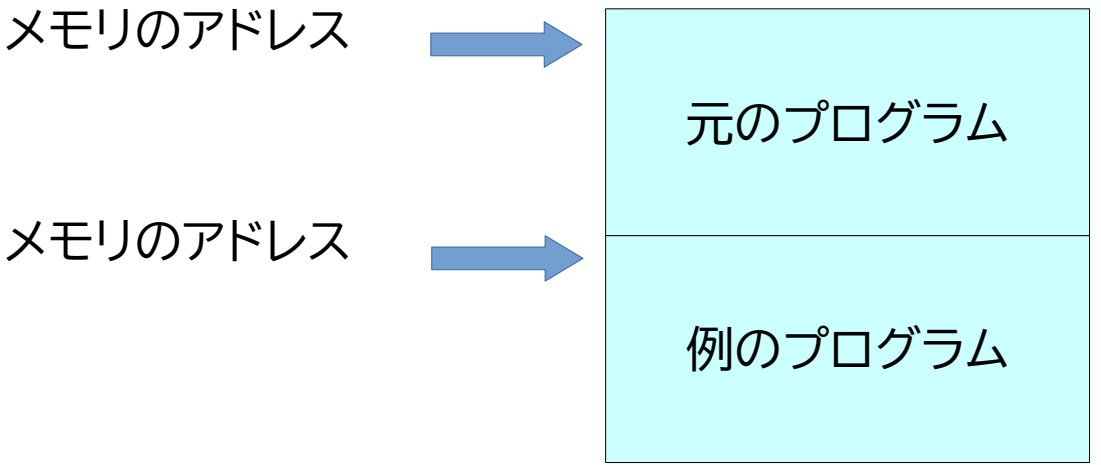
mainはラベル

1 + 1の足し算に相当

$raレジスタのアドレスが指す命令に戻る

addi $t0, $zero, 1
意味 ゼロレジスタの値(つまりは0と)1を足して、$t0レジスタに格納

$addi $t0, $t0 ,1
$t0レジスタの値(つまりは1)と1を足して、$t0レジスタに格納



今日の内容

- 1週目
  - プログラミング環境の構築
  - MIPSアーキテクチャの概要
  - 命令セットの考え方
  - 簡単なプログラミング

MIPSアーキテクチャ

- シンプルな命令の組み合わせでプログラムを実現
- ハードウェア構成が単純になるように設計されている
- 1命令が32ビット（4バイト）で表現する

前回の復習　プログラム内蔵方式

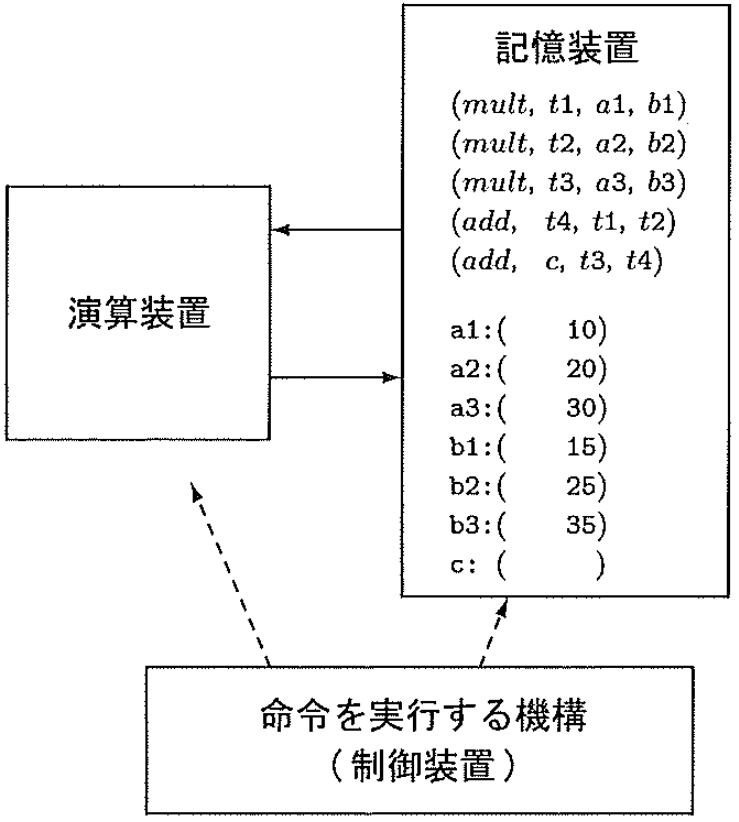


図 6.1　コンピュータの動作(概念図)

復習　プログラム内蔵方式

- ジョン・フォン・ノイマンが提唱した「プログラム内蔵方式」がベースにある。
- ある命令の動作　(op, oprd1, oprd2, oprd3)
- 動作の流れ
- opを解読して実行し、oprd2とoprd3を対象に演算する、その結果をoprd1格納する、継の命令を実行する。

命令の例

- 3次元のベクトル　$a=(a_1,a_2,a_3)$と $b=(b_1,b_2,b_3)$の内積を求める手順を考える

$$c=a_1\cdot b_1+a_2\cdot b_2+a_3\cdot b_3$$

- 計算手順を分解するために、四則演算で考える

命令の例

① 変数 a_1 の値と変数 b_1 の値を乗算して，一時変数 t_1 に代入する．

② 変数 a_2 の値と変数 b_2 の値を乗算して，一時変数 t_2 に代入する．

③ 変数 a_3 の値と変数 b_3 の値を乗算して，一時変数 t_3 に代入する．

④ 一時変数 t_1 の値と一時変数 t_2 の値を加えて，一時変数 t_4 に代入する．

⑤ 一時変数 t_3 の値と一時変数 t_4 の値を加えて，変数 c に代入する．

MIPSアーキテクチャ

- 計算結果を一時的な変数に保存している。コンピュータの命令として形式化するために、
- (op, oprd1, oprd2, oprd3)
- を一組で表記する。
- opは演算、oprdは演算対象のデータ
- 個別の内積要素の計算結果を加算することで、内積が求まる。

命令の例

$(mult, t_1, a_1, b_1)$

$(mult, t_2, a_2, b_2)$

$(mult, t_3, a_3, b_3)$

(add, t_4, t_1, t_2)

(add, c, t_3, t_4)

$*$ $mult$ は乗算，add は加算を表す．

再びこの図　プログラム内蔵方式

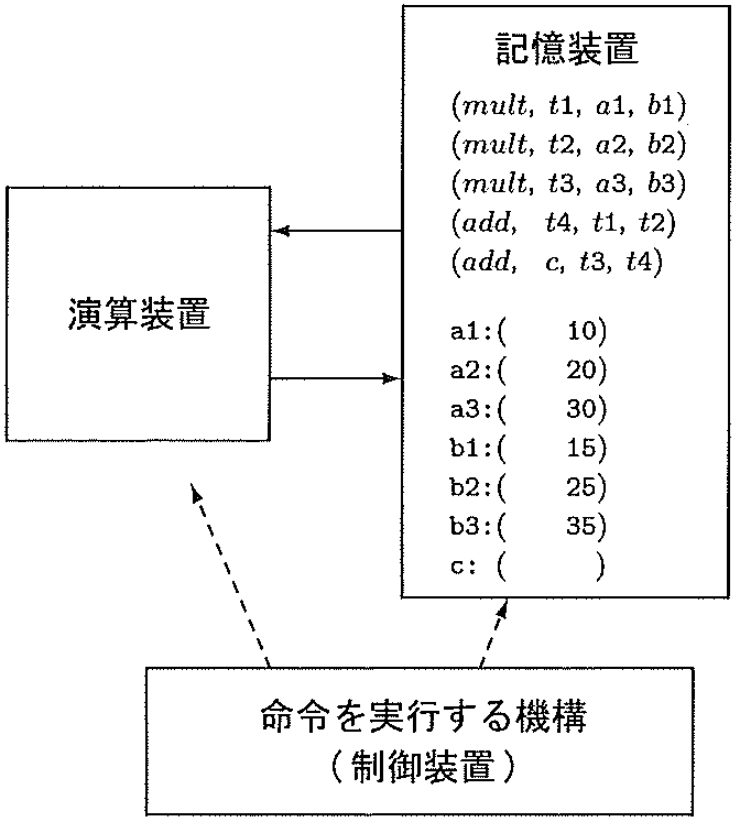


図 6.1　コンピュータの動作(概念図)

コンピュータのメモリ階層

レジスタは演算装置に最も近くてかつ高速(最速)の小容量のメモリ

レジスタ

- レジスタは小容量　32ビット（MIPSの場合）
- 個数のことを「本」と呼ぶ、32本ある
- 目的に応じたさまざまなレジスタがある
- コンパイラは効率よく計算させるためにレジスタを上手に使い分けている

レジスタの番号と種類

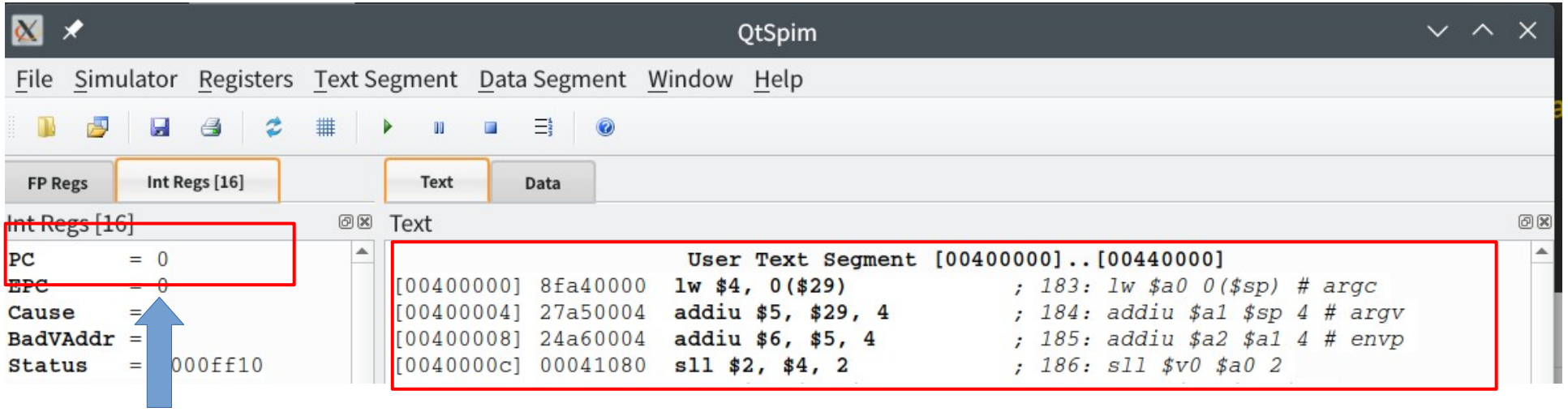
表 6.1　レジスタ番号とレジスタ名

番号	名前	意味
0	$zero	定数 0 をもつ読出し専用レジスタ
1	$at	アセンブラで使用
2〜3	$v0〜$v1	関数の戻り値を格納
4〜7	$a0〜$a3	関数の引数を格納
8〜15, 24, 25	$t0〜$t7, $t8, $t9	一時レジスタ
16〜23	$s0〜$s7	保存レジスタ
26〜27	$k0〜$k1	OS で使用
28	$gp	グローバルポインタ
29	$sp	スタックポインタ
30	$fp	フレームポインタ
31	$ra	戻りアドレス

レジスタの番号と種類

- 教科書p.109

命令とプログラムカウンタ

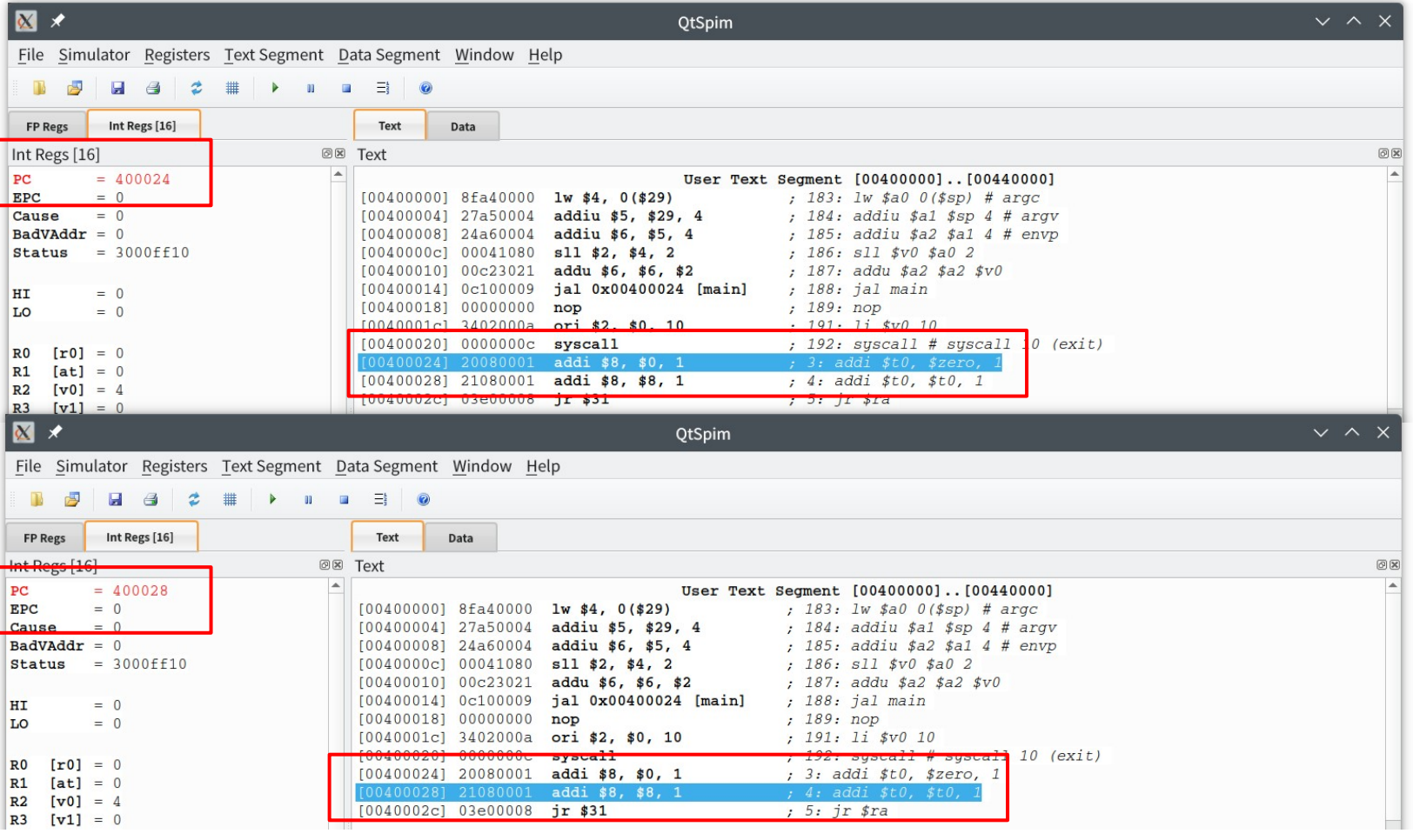


プログラムカウンタ　次の命令を実行するメモリのアドレスを指す

メモリ上の命令

プログラムカウンタと命令

4増えている



プログラムカウンタと命令

- コンピュータが1クロック進むたびにプログラムカウンタは4ずつ(4バイト、1ワード)増える

プログラムカウンタと命令

- 教科書p.252の概念図

プロセッサとメモリの対応関係

- 教科書p.76の概念図

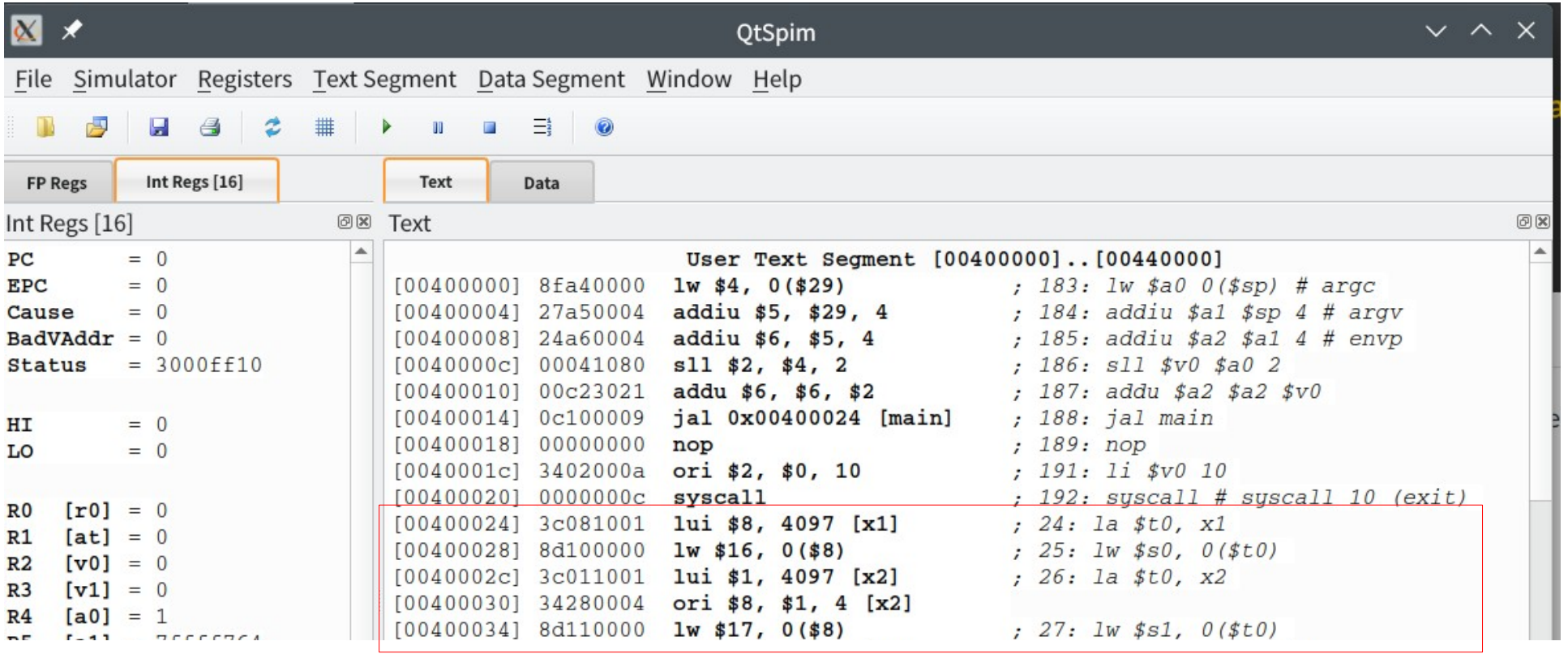
今日の内容

- 1週目
  - プログラミング環境の構築
  - MIPSアーキテクチャの概要
  - 命令セットの考え方
  - 簡単なプログラミング

命令の一覧(実習内で使用するもの)

- 命令セット　教科書p.70、p.201

命令の実際(マシン語とアセンブリ言語)



命令の実際(マシン語とアセンブリ言語)

```
[00400034] 8d110000  lw $17, 0($8)
```

アドレス　　マシン語　　アセンブリ言語

- 命令によってマシン語表現が異なる
- 命令ごとにマシン語が定義されている
- アセンブラ(コンパイラに対応)でマシン語に変換している

命令のフォーマット

- 32ビットの固定長(CPUによっては可変長)
- 基本的な考え方

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6
命令	レジスタ1	レジスタ2	レジスタ3 (保存先)		

この並びとアセンブリ言語の表記は異なるので注意(特にレジスタの順番)

命令のフォーマット

- 命令フォーマットの違いは教科書p.89.
- R形式、I形式、J形式がある
- R形式の命令（レジスタに結果を格納）

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6
命令	レジスタ1	レジスタ2	レジスタ3 （保存先）		

命令のフォーマット

- 命令フォーマットの違いは教科書p.89.
- R形式、I形式、J形式がある
- I形式

op	rs	rt	addressまたはconstant
6	5	5	16
命令	レジスタ1	レジスタ2	

命令のフォーマット

- 命令フォーマットの違いは教科書p.89.
- R形式、I形式、J形式がある
- J形式（命令の遷移に使う）

op	address
6	26

命令

命令とマシン語の例

- add $t0, $s1, $s2

op	rs	rt	rd	shamt	funct
000000	10001	10010	01000	00000	100000

今日の内容

- 1週目
  - プログラミング環境の構築
  - MIPSアーキテクチャの概要
  - 命令セットの考え方
  - 簡単なプログラミング

レポート作成に向けて

- 練習(サンプル)のプログラムを実装する
- 演習のプログラムを実装する
- 演習のプログラムをレジスタとステップを示して実行の様子を表に整理する

加算・減算

- 一時レジスタ、保存レジスタを使用（前スライド）
- 命令の使い方の例は教科書を参照
- add
- sub
- addi
  - addi rd, rs, rt
  - 意味　rd = rs + rt
  - rtの値を指定できる

演習1

1. $t0レジスタに100をセット
2. $t1レジスタに200をセット
3. $t0と$t1の和を$t3に保存

ファイル名：exercise_01.asm

演習2

1. $t0レジスタに100をセット
2. $t1レジスタに-200をセット
3. $t0と$t1の差を$t3に保存

subi命令は用意されていないため、addiで実現する

ファイル名：exercise_02.asm

ロード・ストア

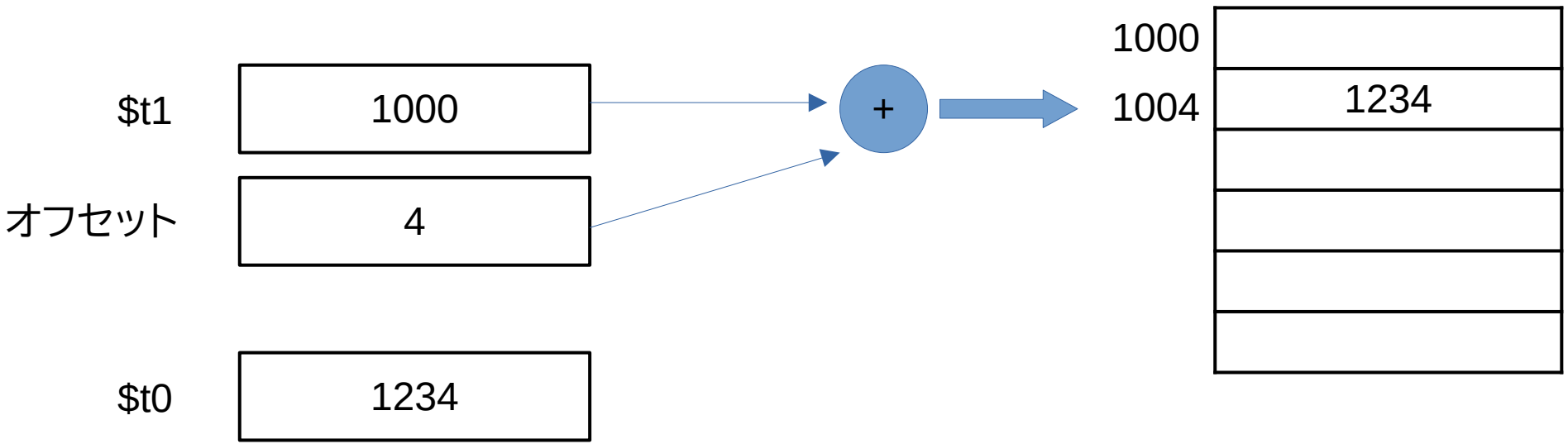
- ロード　メモリに保存されているデータの読み込み
  - lw レジスタ, オフセット(ベースアドレス)
- ストア　指定されたアドレスへのデータの保存
  - sw レジスタ, オフセット(ベースアドレス)

ロードの例1

- 例 lw $t0, 0($t1)
  - $t1レジスタの値にオフセットの「0」を加算したアドレスが指すメモリ上のデータを読み込み$t0レジスタに保存する

相対アドレスの考え方

- 例 lw $t0, 4($t1)

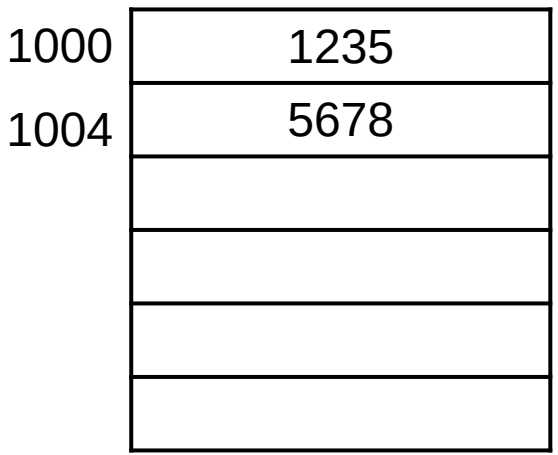


データ領域の定義

```
        .data

        .globl v
v:      .word   1234
        .word   5678

        .text

        .globl  main
main:
        la $t3, v
        lw $t0, 0($t3)
        lw $t1, 4($t3)
        jr $ra
```

データ領域開始

データ領域終了



ストアの例

- 例 sw $t0, 0($t1)
  - $t1レジスタの値にオフセットの「0」を加算したアドレスが指すメモリ上のアドレスへ、$t0レジスタのデータを保存

ロードの例2(オフセットあり)

- 例 lw $t0, 4($t1)
  - $t1レジスタの値にオフセットの「4」を加算したアドレスが指すメモリ上のデータを読み込み$t0レジスタに保存する

ストアの例2(オフセットあり)

- 例 sw $t0, 4($t1)
  - $t1レジスタの値にオフセットの「4」を加算したアドレスが指すメモリ上のアドレスへ、$t0レジスタのデータを保存

演習3

- 4ワード分のデータ領域を用意し、その中に数値を保存する（値は適当でよい）。1ワード目のデータを4ワード目、3ワード目のデータを2ワード目にコピーするプログラムを作成する。

```
        .data

        .globl v
v:      .word   1234
        .word   5678

        .text

        .globl  main
main:
        la $t3, v
        lw $t0, 0($t3)
        lw $t1, 4($t3)
        jr $ra
```

ファイル名：exercise_03.asm

演習課題のファイル名

- 以後の課題のファイル名　exercise_XX.asm
- XXは課題の番号

論理演算

- and
- or
- nor

演習4

- not演算に相当する処理をするプログラムを作成する。$t0レジスタの値を$t1レジスタにnot演算を施した結果を保存する。

シフト演算

- sll
- srl

演習5

- シフト演算対象の数値を$t0レジスタに保存し、$t0の値を16倍して、$結果を$t2レジスタに保存する。$t0の値を1/4倍して$t3レジスタに保存するプログラムを作成する。

比較

- slt
- slti
- sltu

演習6

- 比較対象の数値を$t0、$t1、$t2レジスタに保存する。$t0 > $t1 かつ$t0 > $t2が成立する例を考えて、比較結果を$t3レジスタに保存するプログラムを作成する。この2つの関係が満たされた回数を$t3に保存する。

条件分岐

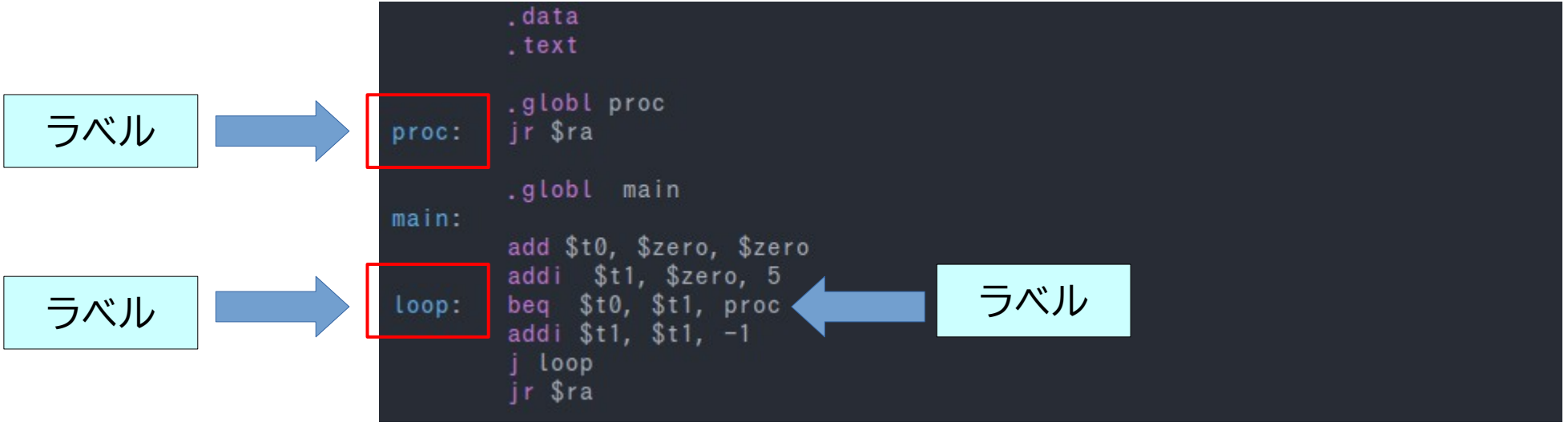
- bne
- beq

演習7

- 比較対象の数値を$t0、$t1に保存する。$t0 < $t1の場合は、$t1の値を1000加算するプログラムを作成する。

繰り返し処理

- 比較命令、加算命令、ラベル等を使って実現する



演習8

- 繰り返し処理を実装して0から5の数字を$t3レジスタに順番に格納するプログラムを作成する。

演習9（演習8の後に取り組む）

- シフト演算対象の数値を$t0レジスタに保存し、$t1レジスタで指定されたビット数だけ右シフトする。結果を$t2レジスタに保存するプログラムを作成する。

演習10

- C言語プログラムの二重ループに相当するプログラムを作成する。下記のプログラムをアセンブリ言語で実装する。プログラム実行終了時のvの値は$t8レジスタに保存する。

```c
#include <stido.h>

int main(void) {
    int i = 0;
    int j = 0;
    int v = 0;
    int m = 2;
    int n = 2;

    for (i = 0; i <= m; i++) {
        for (j = 0; j <= n; j++) {
            v = v + 1;
        }
    }
}
```