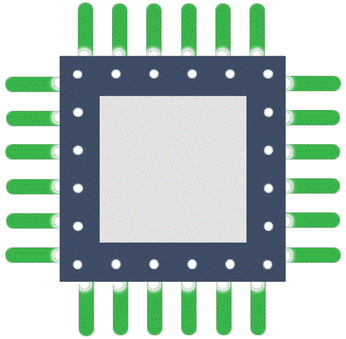
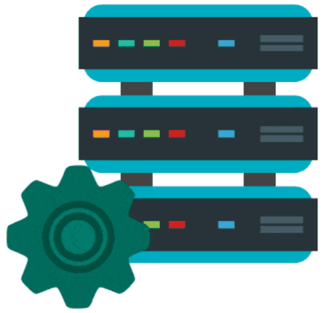


MIPS

情報領域演習第二 C演習 第２回

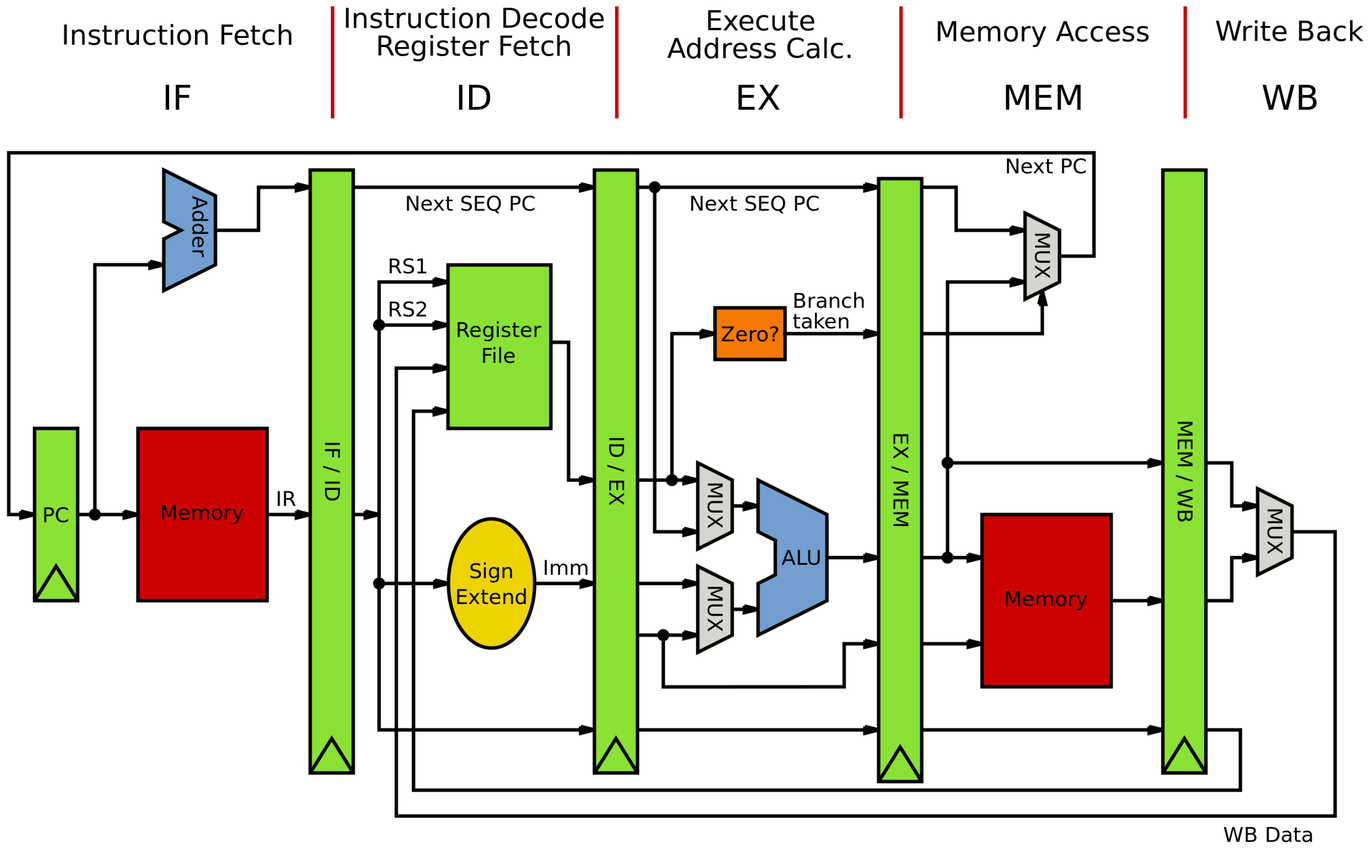
クラス２　2025年5月23日1限　担当教員：高木一幸

出席カードに学籍番号と氏名を記入し、
前方中央（2つのスクリーンの中間）の
缶に提出。
特に数字は標準字体で
丁寧に記入すること。



MIPS アセンブリ言語はハードウェア操作用言語

MIPSアーキテクチャを動作させるためのものなので...



- ✳ レジスタの用途、種類、数は固定
  - ✳ 任意の名前、番号のレジスタは使用できない
  - ✳ 用途外の使い方をしてはならない
- ✳ 演算、シフト、データ転送、分岐、サブルーチン呼び出しなどの命令に使えるレジスタの種類、数、順番は固定
  - ✳ 構文と異なるレジスタは使えない
  - ✳ 複雑な計算は複数命令に分割実施

e.g. 「x ← a + b + c」

→ x ← a + b

x ← x + c

https://ja.wikipedia.org/wiki/MIPSアーキテクチャ#/media/ファイル:MIPS_Architecture_(Pipelined).svg

MIPSプログラムの基本形

```
.data   # ここから先はデータセグメント （データがなければ不要）
        ここにデータを記載する
```

```
.text                       # ここから先はテキストセグメント
.globl main                 # main をグローバルラベルとする
main: addi $sp, $sp, -4     # スタック領域の確保（第3回で解説）
      sw $ra, 0($sp)        # 戻り先アドレスの退避（第3回で解説）
```

```
      ここに計算の本体を書く
```

```
      lw $ra, 0($sp)    # 戻り先アドレスの復元 （第3回で解説）
      addi $sp, $sp, 4  # スタック領域の開放    （第3回で解説）
      jr $ra            # main を終了してリターンする
```

プログラム例：1から10の総和

```
    .text                           # ここから先はテキストセグメント
    .globl  main                    # mainをグローバルラベルとする
main: addi  $sp, $sp, -4            # スタック領域の確保（第3回で解説）
      sw    $ra, 0($sp)             # 戻り先アドレスの退避（第3回で解説）
      move  $t0, $zero              # $t0 ← $zero
      li    $t1, 10                 # $t1 ← 10
loop: add   $t0, $t0, $t1           # $t0 ← $t0+$t1
      sub   $t1, $t1, 1             # $t1 ← $t1-1
      bne   $t1, $zero, loop        # 非0ならloopへ
      li    $v0, 1                  # print_intのシステムコール
      move  $a0, $t0                # プリントする整数を$a0に格納
      syscall                       # システムコール
      lw    $ra, 0($sp)             # 戻り先アドレスの復元（第3回で解説）
      addi  $sp, $sp, 4             # スタック領域の開放（第3回で解説）
      jr    $ra                     # mainを終了してリターン
```

対応

プログラム例：1から10の総和

```
        .text                       # ここから先はテキストセグメント
        .globl  main                # mainをグローバルラベルとする
main:   addi    $sp, $sp, -4        # スタック領域の確保（第3回で解説）
        sw      $ra, 0($sp)         # 戻り先アドレスの退避（第3回で解説）
        move    $t0, $zero          # $t0 ← $zero
        li      $t1, 10             # $t1 ← 10
loop:   add     $t0, $t0, $t1       # $t0 ← $t0+$t1
        sub     $t1, $t1, 1         # $t1 ← $t1-1
        bne     $t1, $zero, loop    # 非0ならloopへ
        li      $v0, 1              # print_intのシステムコール
        move    $a0, $t0            # プリントする整数を$a0に格納
        syscall                     # システムコール
        lw      $ra, 0($sp)         # 戻り先アドレスの復元（第3回で解説）
        addi    $sp, $sp, 4         # スタック領域の開放（第3回で解説）
        jr      $ra                 # mainを終了してリターン
```

プログラム例：整数の和

```
    .data                                   # ここから先はデータセグメント
prmpt1:     .asciiz     "1st integer = "    # 第１整数の入力のプロンプト文字列
prmpt2:     .asciiz     "2nd integer = "    # 第２整数の入力のプロンプト文字列
ans:        .asciiz     "sum = "            # ２整数の和を示す文字列
newln:      .asciiz     "\n"                # 改行文字

    .text                                   # ここから先はテキストセグメント
    .globl  main                            # mainをグローバルラベルとする

main: addi  $sp, $sp, -4                    # スタック領域の確保（第3回で解説）
      sw    $ra, 0($sp)                     # 戻り先アドレスの退避（第3回で解説）
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, prmpt1                     # 第１整数の入力のプロンプト文字列のアドレスを$a0に設定
      syscall                               # プロンプトの表示
      li    $v0, 5                          # read_intのシステムコールコード
      syscall                               # 整数値を読み込む
      move  $t1,$v0                         # 読み込んだ整数値$v0を$t1に設定
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, prmpt2                     # 第２整数の入力のプロンプト文字列のアドレスを設定
      syscall                               # プロンプトの表示
      li    $v0, 5                          # read_intのシステムコールコード
      syscall                               # 端末から整数値を読み込む
      move  $t2,$v0                         # 読み込んだ整数値$v0を$t2に設定
      add   $t3,$t1,$2                      # $t3 ← $t1+$t2
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, ans                        # ２整数の和を示す文字列のアドレスを$a0に設定
      syscall                               # 文字列を表示
      li    $v0, 1                          # print_intのシステムコールコード
      move  $a0, $t3                        # $t3（２整数の和）を$a0に設定
      syscall                               # 整数値を表示
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, newln                      # 改行文字のアドレスを設定
      syscall                               # 改行文字を表示
      lw    $ra, 0($sp)                     # 戻り先アドレスの復元 （第3回で解説）
      addi  $sp, $sp, 4                     # スタック領域の開放 （第3回で解説）
      jr    $ra                             # mainを終了してリターン
```

対応

実行例

```
(spim) run

1st integer = 7

2nd integer = 19

sum = 26

(spim)
```

プログラム例：整数の和

```
    .data                                   # ここから先はデータセグメント
prmpt1:     .asciiz     "1st integer = "    # 第１整数の入力のプロンプト文字列
prmpt2:     .asciiz     "2nd integer = "    # 第２整数の入力のプロンプト文字列
ans:        .asciiz     "sum = "            # ２整数の和を示す文字列
newln:      .asciiz     "\n"                # 改行文字

    .text                                   # ここから先はテキストセグメント
    .globl  main                            # mainをグローバルラベルとする

main: addi  $sp, $sp, -4                    # スタック領域の確保（第3回で解説）
      sw    $ra, 0($sp)                     # 戻り先アドレスの退避（第3回で解説）
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, prmpt1                     # 第１整数の入力のプロンプト文字列のアドレスを$a0に設定
      syscall                               # プロンプトの表示
      li    $v0, 5                          # read_intのシステムコールコード
      syscall                               # 整数値を読み込む
      move  $t1,$v0                         # 読み込んだ整数値$v0を$t1に設定
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, prmpt2                     # 第２整数の入力のプロンプト文字列のアドレスを設定
      syscall                               # プロンプトの表示
      li    $v0, 5                          # read_intのシステムコールコード
      syscall                               # 端末から整数値を読み込む
      move  $t2,$v0                         # 読み込んだ整数値$v0を$t2に設定
      add   $t3,$t1,$2                      # $t3 ← $t1+$t2
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, ans                        # ２整数の和を示す文字列のアドレスを$a0に設定
      syscall                               # 文字列を表示
      li    $v0, 1                          # print_intのシステムコールコード
      move  $a0, $t3                        # $t3（２整数の和）を$a0に設定
      syscall                               # 整数値を表示
      li    $v0, 4                          # print_stringのシステムコールコード
      la    $a0, newln                      # 改行文字のアドレスを設定
      syscall                               # 改行文字を表示
      lw    $ra, 0($sp)                     # 戻り先アドレスの復元 （第3回で解説）
      addi  $sp, $sp, 4                     # スタック領域の開放 （第3回で解説）
      jr    $ra                             # mainを終了してリターン
```

実行例

```
(spim) run

1st integer = 7

2nd integer = 19

sum = 26

(spim)
```

reinitialize

メモリとレジスターの初期化

ファイルが既にSPIMに読み込まれている状態で、
新たにファイルを読み込む場合、
SPIMを初期化しなければならない。
さもなくば、global記号（mainなど）が重複して
定義されることになり、エラーが生じる。

reinitialize（再初期化）コマンドによってメモリと
レジスタを初期化する必要がある。

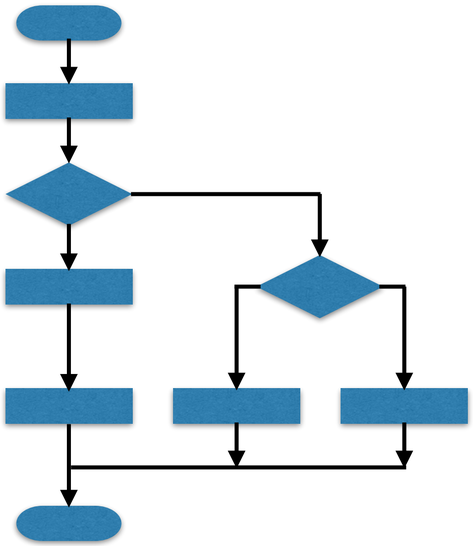
```
[ta103027@yellow49 ~/spim]$ spim
Loaded: /usr/share/spim/exceptions.s
(spim) load "26Jun2020/sum1to10_2.s"
(spim) run
55(spim) reinitialize
Loaded: /usr/share/spim/exceptions.s
SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
(spim) load "sumint.s"
(spim) run
1st integer = 7
2nd integer = 19
sum = 26
(spim)
```

実行例の作り方

✳SPIM起動直後のレジスタの初期値に依存しないプログラムであること。このことを示すため、複数の入力例を SPIM を exit（あるいは quit ）せずに連続して実行すること。

✳キーボード入力を伴う課題では入力や条件が異なる実行例を示すこと。

- ✳ 扱う値の範囲の入力例を残らず試す（例）整数は自然数に0と負の数を加えた数
- ✳ プログラムの条件分岐経路を全て通過する
- ✳ 数列では、漸化式のすべての条件を実行する
- ✳ プログラムの機能を示す必要最小限の実行例に留める



✳特に指定がない限り、処理結果が扱うデータ型の値の範囲に収まると仮定してよい。

レポート課題

期限は2025年5月29日23:59:59

確認 →

課題１

キーボードから読み込んだ整数値を**x**とするとき、**93x+5**を計算。

3種類のアルゴリズムで。①加算の繰り返し（ループを形成） ②乗算　③左シフトと加算の組み合わせ

課題２

キーボードから読み込んだ整数値が 8の倍数の場合は ”**yes**”、

さもなくば ”**no**” を表示。論理演算を用いること。

課題３

Pell数列を初項から第***n***項まで表示する。項数***n***はキーボードから入力すること

$$P(1) = 1, \quad P(2) = 2, \quad P(n) = 2P(n-1) + P(n-2) \ \ (n \geqq 3)$$

詳細 →

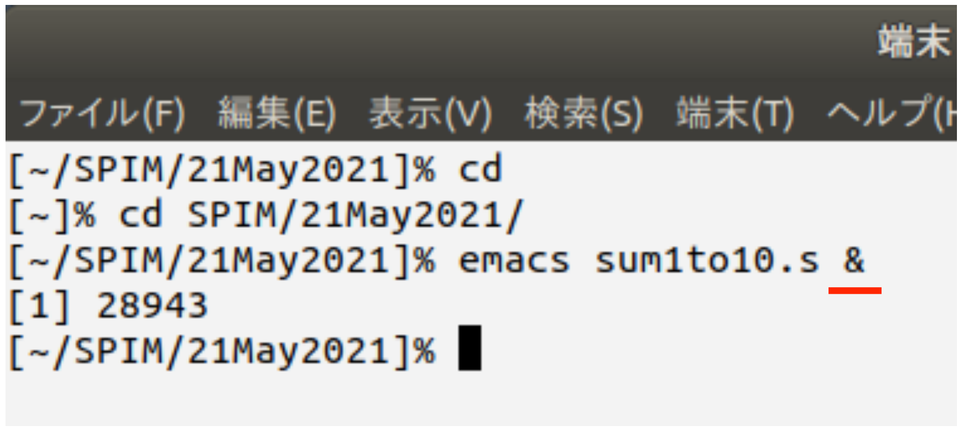
MIPSプログラミング作業

ubuntu を起動。

端末を起動。

端末を複数並べて使う

emacsなどを起動。端末からCUIで起動する場合には　必ず "&" をつける



（例）ファイル **sum1to10.s** を指定してemacsを起動。

デスクトップは分割して有効に使う

端末やアプリケーションを最大化しない

✳ SPIMは端末で実行

✳ プログラムの編集はemacsなどのプログラミング環境・エディタで