

計算機工学実験I (第12~15回)

機械語・アセンブリ言語

森本

1

機械語とアセンブリ言語

機械語

- ・計算機が扱える0と1の羅列

00000000110000100011000000100001

↑
↓ ほぼ1対1の対応

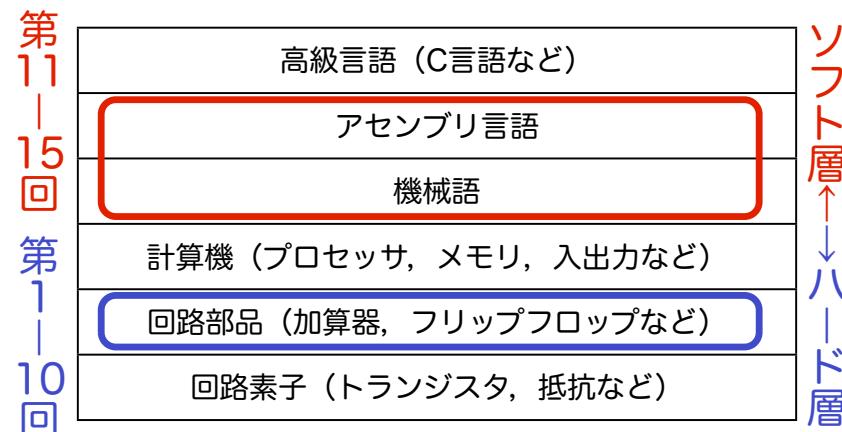
addu \$a2, \$a2, \$v0

アセンブリ言語

- ・機械語を人間向けに読みやすくしたもの

3

実験の全体像



1

Hello World

機械語

3

Hello World

アセンブリ言語

```
.data  
str: .asciiiz "Hello, World\n"  
.text  
.align 2  
.globl main  
  
main:  
    li      $v0,4  
    la      $a0,str  
    syscall  
    move   $v0,$0  
    j       $ra
```

5

Hello World

ラベル 命令 オペランド

ラベル	命令	オペランド
	.data	
str:	.asciiiz	"Hello, World\n"
	.text	
	.align	2
	.globl	main
main	li	\$v0,4
	la	\$a0,str
	syscall	
	move	\$v0,\$0
	j	\$ra

6

Hello World

データ領域

```
.data  
str: .asciiiz "Hello, World\n"  
  
.text  
.align 2  
.globl main  
  
main:  
    li      $v0,4  
    la      $a0,str  
    syscall  
    move   $v0,$0  
    j       $ra
```

プログラム領域

静的データ領域

実行前に確保

プログラム領域

```
.data  
str: .asciiiz "Hello, World\n"  
  
.text  
.align 2  
.globl main  
  
main:  
    li      $v0,4  
    la      $a0,str  
    syscall  
    move   $v0,$0  
    j       $ra
```

7

8

Hello World

プログラム全体の
入口

出口
(return 0)

```
.data
str: .asciiiz "Hello, World\n"
.text
.align 2
.globl main
main:
    li      $v0,4
    la      $a0,str
    syscall
    move   $v0,$0
    j       $ra
```

Hello World

文字列の表示

```
.data
str: .asciiiz "Hello, World\n"
.text
main:
    li      $v0,4
    la      $a0,str
    syscall
    move   $v0,$0
    j       $ra
```

文字列表示用システムコール
を意味する番号 4 を設定

表示する文字列を設定

システムコール

9

10

実験の対象と環境

11

MIPS

Microprocessor without Interlocked Pipeline Stages

- MIPS Technologies 社のプロセッサ
- パイプラインの流れが乱れにくい
- RISC (Reduced Instruction Set Computer)
- ゲーム機、組み込み機器等で広く採用
 - NINTENDO64, PS, PS2, PSP
 - Windows CE 製品
 - Ciscoのルータ

12

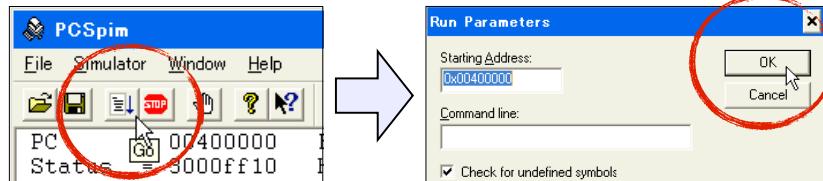
SPIM

- MIPSのシミュレータ
 - MIPS用アセンブリプログラムを解釈実行
- 様々なOSに対応
 - Windows, UNIX, Mac OS
 - 自宅でも手軽に予復習できる
 - <http://www.cs.wisc.edu/~larus/spim.html>

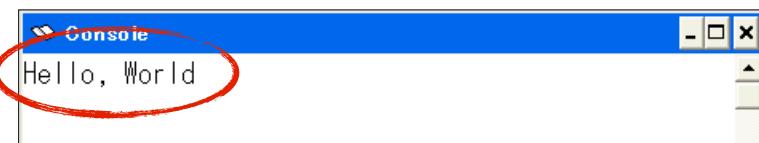
13

SPIMの使い方

3. 読み込んだプログラムを実行



4. 実行結果がコンソールに表示される



画面構成

レジスタ

```
PC = 00000000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000000
Status = 3000ff10 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 10010000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
```

プログラム領域

```
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 175: lw $a0 0($sp)
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $v0 $a1 4
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
[0x00400010] 0x0cc23021 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
[0x00400014] 0x0c10009 jal 0x0400024 [main] ; 180: jal main
[0x00400018] 0x00000000 nop ; 181: nop
[0x0040001c] 0x340200a ori $2, $0, 10 ; 183: li $v0 10
```

データ領域

```
DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x2c6c6548 0x57202c6f 0x646c726f 0x0000000a
[0x10010010]...[0x10040000] 0x00000000

STACK
[0xffffef68] 0x00000000 0x00000000
[0xffffef70] 0x7fffffc9 0x7ffffef92 0x7ffffef7f 0x7ffffef4e
```

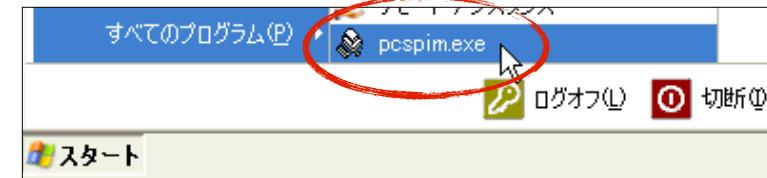
メッセージ

```
SPIM Version 7.5 of August 14, 2009
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Program Files\PCSpim\exceptions.s
Memory and registers cleared and the simulator reinitialized.
```

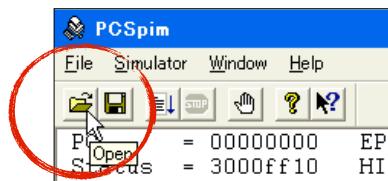
14

SPIMの使い方

1. Windows版SPIM「pcspim.exe」を起動



2. 自作の機械語プログラムを読み込む



- ・プログラム作成には、お好みのエディタを用いる。
- ・拡張子は必ず「.s」にしておく。

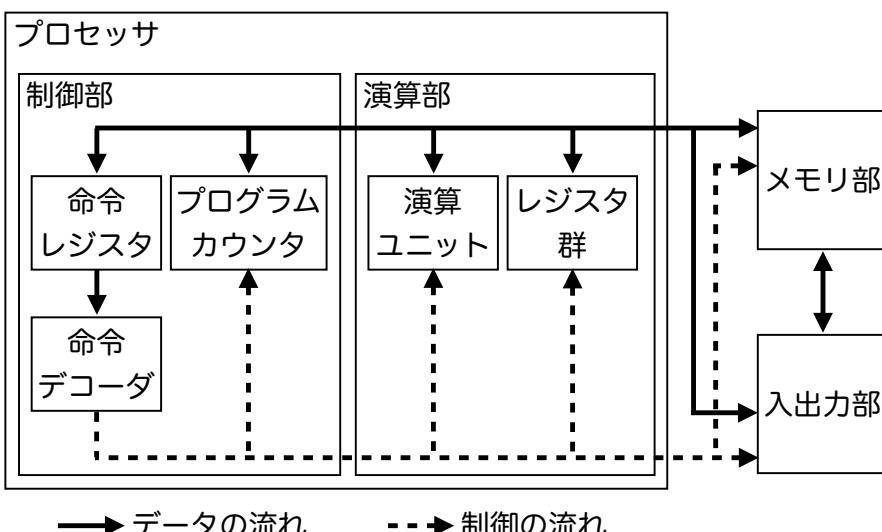
15

課題B1 【必修, 10点】

- "Hello, World" を表示するプログラムを作成し, SPIM上で実行せよ
- 自己の学生番号を表示するプログラムを作成・実行せよ
- レポートには以下を記入せよ
 - プログラムリスト (学生番号表示版のみ)
 - 実行手順 (SPIMの使用方法)
 - 実行結果 (コンソールの表示内容)

17

計算機の構成



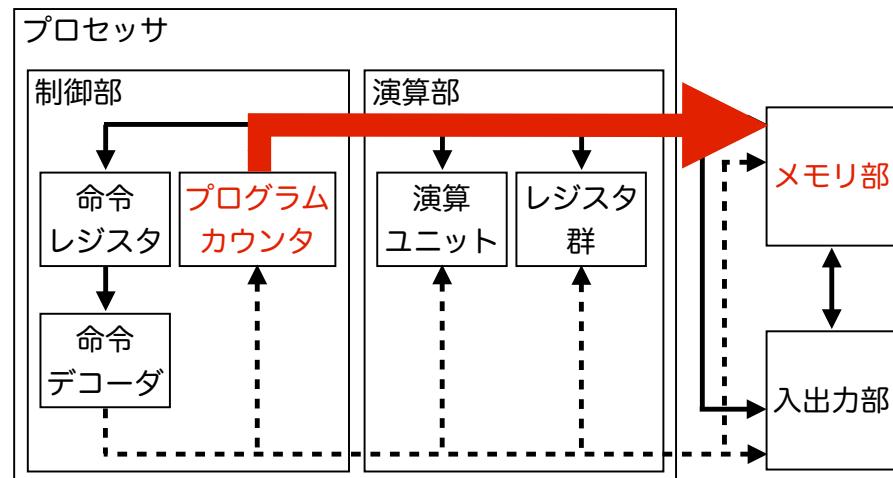
19

【復習】

計算機と機械語

プログラムの実行の流れ

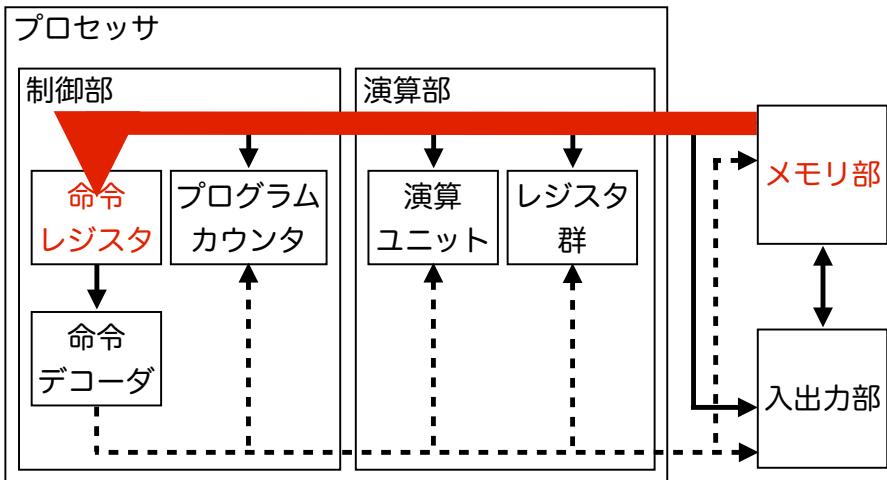
- ① 次に実行する機械語命令を, メモリから読み



20

プログラムの実行の流れ

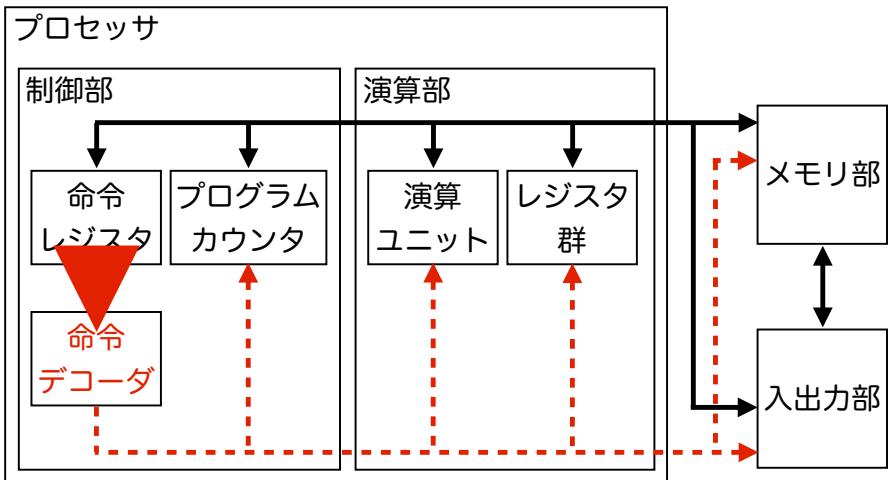
② その機械語命令を、命令レジスタに格納する



21

プログラムの実行の流れ

③ 機械語命令を解釈し、制御信号を各部に送信



22

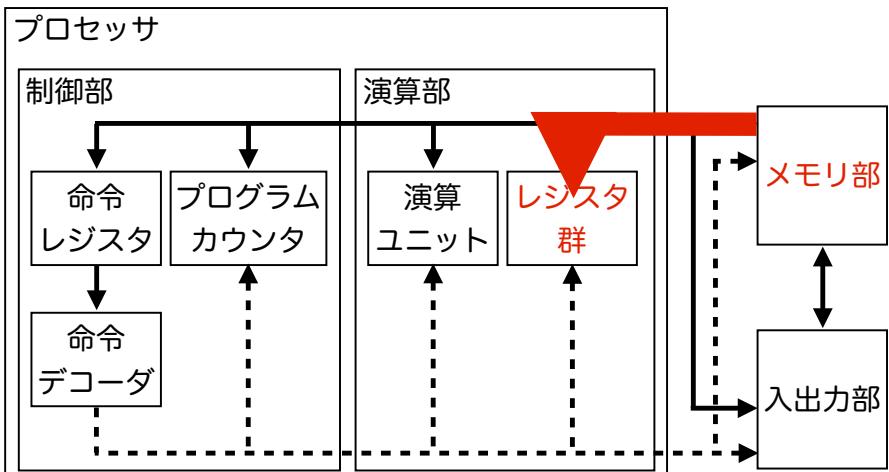
主な機械語命令

- データ転送命令
 - ロード, ストア, ムーブ
- 演算命令
 - 四則, 論理, シフト
- 比較命令
- 分岐命令
 - 無条件ジャンプ, 条件分岐
- システムコール
 - オペレーティングシステムの機能を呼び出す

23

ロード命令

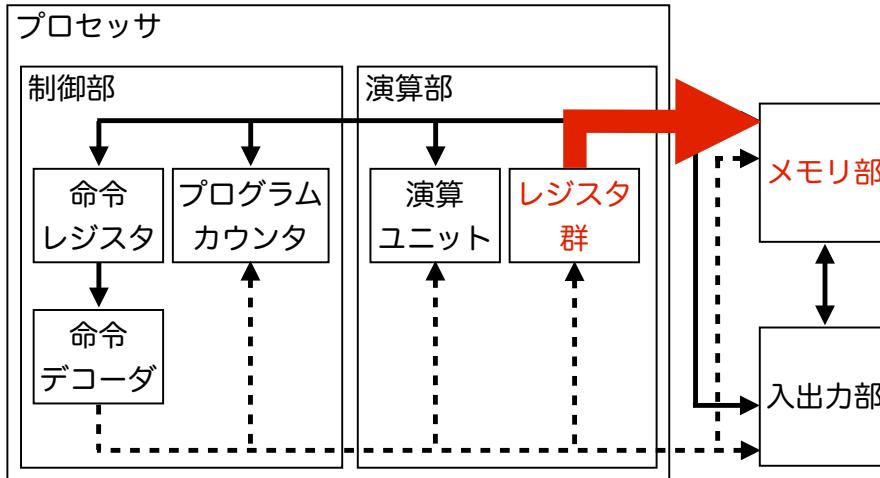
メモリ上のデータを、レジスタに転送



24

ストア命令

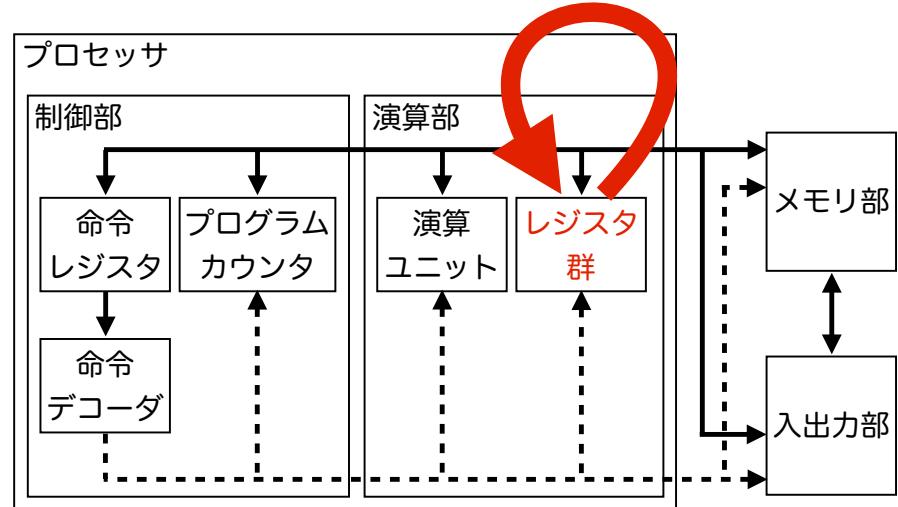
レジスタのデータを、メモリに転送



25

ムーブ命令

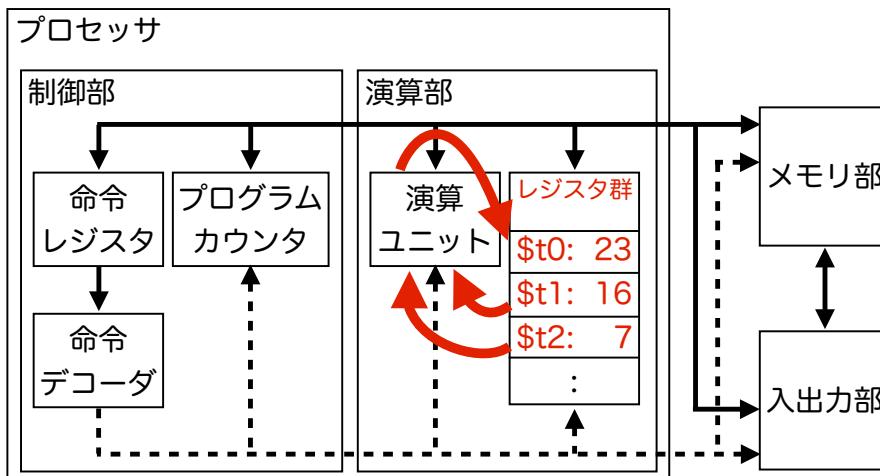
レジスタのデータを、別のレジスタに転送



26

演算系や比較系の命令

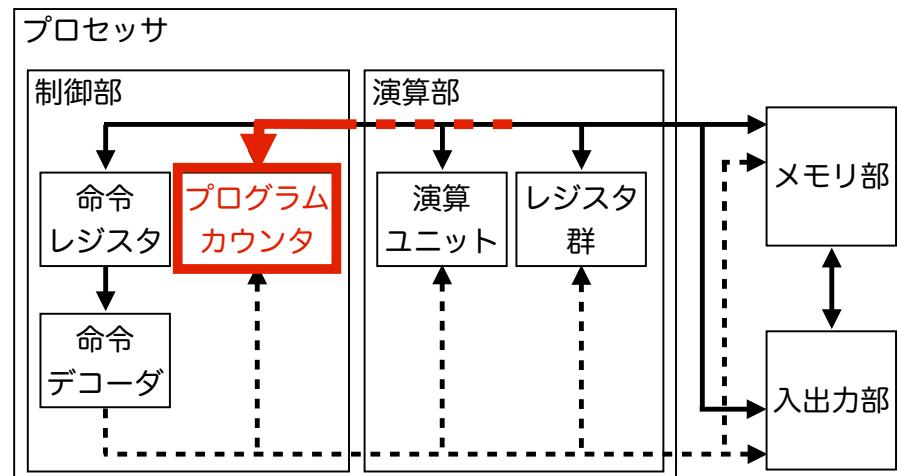
【加算の例】 addu \$t0, \$t1, \$t2



27

分岐命令

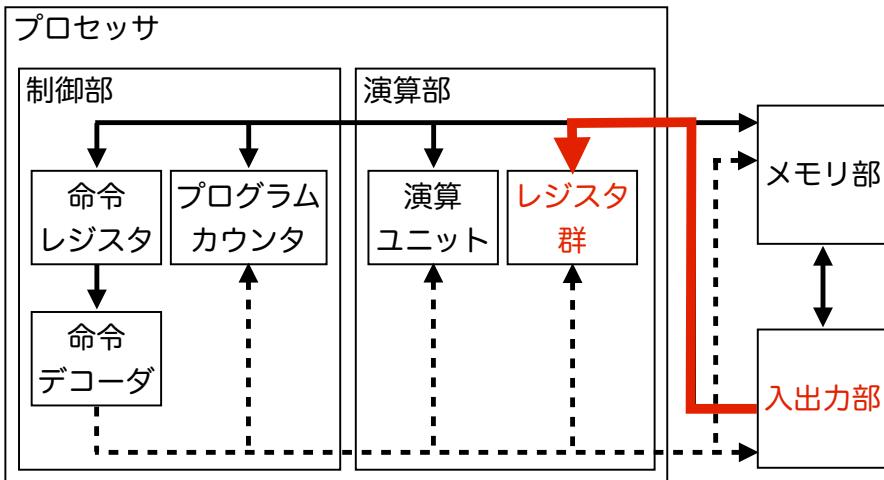
プログラムカウンタの内容を変更



28

システムコール

例えば、キーボードの入力をレジスタに転送



29

MIPSに固有の事柄

レジスタの用途と別名

本名	別名	用途
\$0	\$zero	常に0が入っている
\$1	\$at	アセンブラー用
\$2	\$v0	返り値 (下位32bit)
\$3	\$v1	返り値 (上位32bit)
\$a0 : \$7	\$a0 : \$a3	引数 (サブルーチンや システムコール時)
\$8 : \$15	\$t0 : \$t7	自由に使ってよいが、 サブルーチン呼び出し 後の値は不定 (呼ぶ側による保存が必要)

本名	別名	用途
\$16 : \$23	\$s0 : \$s7	自由に使ってよいし、 サブルーチン呼び出し後 も値は保存される (呼ばれる側による保存が必要)
\$24	\$t8	位置独立コードの
\$25	\$t9	間接ジャンプ用
\$26 \$27	\$k0 \$k1	カーネル用
\$28	\$gp	静的データ領域の起点
\$29	\$sp	Stack Pointer
\$30	\$fp	Frame Pointer
\$31	\$ra	サブルーチンの戻り番地

31

システムコール

番号	機能	引数	返り値
1	intを表示	\$a0	
2	floatを表示	\$f12 (浮動小数点数用のレジスタ)	
3	doubleを表示	\$f12 (浮動小数点数用のレジスタ)	
4	文字列を表示	\$a0	
5	intを入力		\$v0 (int)
6	floatを入力		\$f0 (float)
7	doubleを入力		\$f0 (double)
8	文字列を入力	\$a0=buffer, \$a1=length	
9	メモリ動的確保	\$a0=amount	\$v0 (address)
10	exit		

32

システムコールの使用例

文字列を表示する

```
.data
str: .asciiiz "Hello, World\n"
.text
main:
    li      $v0,4      表示する文字列str
    la      $a0,str     を$a0に設定
    syscall          システムコール
    move   $v0,$0
    j       $ra
```

文字列表示システムコールを意味する番号4を\$v0に設定

33

システムコールの使用例

キーボードから入力した整数を、画面に表示

```
li      $v0,5      # 整数入力システムコールの番号5を設定
syscall          # 入力された整数が$v0に入る

move  $a0,$v0  # その整数をシステムコールの引数にする
li      $v0,1      # 整数表示システムコールの番号1を設定
syscall          # 入力された整数が画面に表示される
```

'#' 以降はコメントになる

34

MIPSの詳細

- このスライドでは詳細は網羅していない

- 適宜、巻末のリファレンス

MIPS/SPIM Reference Card

[Jan Wätzig, 2007]

を参照すること

35

C言語との対応付け

36

メモリの静的確保

C言語

```
char *msg = "Hello, World\n";
int array[ ] = { 0, 1, 2 };
```

アセンブリ言語

```
.data
msg: .asciiz "Hello, World\n"
array: .word 0, 1, 2
```

- メモリを静的に確保するためのアセンブリ指令
 - `.asciiz` ← 文字列用のメモリを確保
 - `.word` ← 32bit単位でメモリを確保

37

定数の代入

C言語

```
int i = 72;
int *p = array;
```

アセンブリ言語

```
li $t1, 72
la $t3, array
```

- レジスタに定数を設定するための命令
 - `li` レジスタ, 即値
 - `la` レジスタ, アドレス

なお, 即値が0の場合は, \$0をムーブする方が高速
(`li $t1, 0`) (`move $t1, $0`)

38

メモリからの読み込み

C言語

```
char c = *msg;
char d = *(msg+1);
int x = array[0];
int y = array[1];
```

アセンブリ言語

```
la $t0, msg
lb $t1, 0($t0)
lb $t2, 1($t0)
la $t3, array
lw $t4, 0($t3)
lw $t5, 4($t3)
```

- メモリ上のデータをレジスタに転送する命令
 - `lb` レジスタ, 符付(レジスタ) ← byte単位
 - `lw` レジスタ, 符付(レジスタ) ← word単位

39

加減算

C言語

```
int x = 123;
int y;
y = x + x;
y = x + 456;
y = x - 789;
```

アセンブリ言語

```
li $t0, 123
addu $t1, $t0, $t0
addu $t1, $t0, 456
subu $t1, $t0, 789
```

- 加減算を行う命令
 - `addu` レジスタ, レジスタ, レジスタ
 - `addu` レジスタ, レジスタ, 即値
 - `subu` も同様

40

繰り返し

C言語	アセンブリ言語
int i = 3; do i -- ; while (i != 0);	li \$t0, 3 loop: subu \$t0, \$t0, 1 bne \$t0, \$0, loop

- 繰り返しは、条件分岐命令を用いて実現する
 - **bne** レジスタA, レジスタB, アドレス
↑ レジスタAとレジスタBが同じ値でなければ、
アドレスに分岐する。 (同じ値ならばそのまま次へ)

41

関数の返り値

C言語	アセンブリ言語	
return 0;	move \$v0, \$0 j \$ra	返り値を \$v0 に設定 呼出し元の番地 \$ra にジャンプ

※ return の実現方法については、
後述の「サブルーチン」で詳しく学ぶ

課題B2 【必修, 10点】

- 配列の和を求めるアセンブリ・プログラムを作成せよ
- まず、次ページのC言語のプログラムを理解せよ
- 次に、次々ページの空欄を埋めて、完成させよ
- レポートには以下を記入せよ (以後の全課題も同様)
 - C言語版プログラムのリスト
 - 完成したプログラムのリスト (コメントも記入)
 - プログラムの流れ図 (PAD等でもよい)
and/or
プログラムの説明 (200文字以上)
 - 実行結果、および、その実行結果の正しさの説明

43

配列の和を表示【C言語】

```
#include <stdio.h>
int array[] = { 1, 2, 4, 8, 16, 32, 64, 128 };
int main() {
    int sum = 0;
    int *a = array;
    int i = 8;
    do {
        int x = *a;
        sum = sum + x;
        a++;
        i--;
    } while (i != 0);
    printf("%d", sum);
    return 0;
}
```

44

配列の和を表示【アセンブリ言語】

```
.data  
array: [ ] # int array[] = {1,2,4,...}  
.text  
.align 2  
.globl main  
  
main:  
    move $a0,[ ] # sum = 0  
    la $t0,[ ] # a = array  
    li $t1,[ ] # i = 8  
  
loop:  
    lw $t2,[ ] # x = *a  
    addu $a0,[ ] # sum = sum + x  
    addu $t0,[ ] # a ++  
    subu $t1,[ ] # i --  
    bne $t1,[ ] # do ... while (i != 0)  
    li [ ] # printf("%d", sum)  
    [ ]  
    move [ ] # return 0  
    j [ ]
```

45

課題B3 【選択, 5点】

- 標準入力（キーボード）から入力された非負整数を、オウム返しにそのまま表示するプログラムを作成せよ
- 入力が非負である限り入出力を繰り返し、負数が入力されたらプログラムを終了するようにせよ
- レポートへの記入事項は課題B2と同様
- 以下のような実行結果が得られることが期待される

> 3	3 を入力
3	3 を出力
> 3	3 を入力
3	3 を出力
> 7	7 を入力
7	7 を出力
> -1	負数を入力
0 : 0	分布を出力
1 : 0	
2 : 0	
3 : 2	
4 : 0	
5 : 0	
6 : 0	
7 : 1	
9 : 0	

46

課題B4 【選択, 5点】

- 課題B4のプログラムを拡張して、入力された非負整数の頻度分布を表示するプログラムを作成せよ
- 簡単のため、非負整数の範囲は0~9のみに限定してよい
- 右のような実行結果を得られることが期待される

> 3	3 を入力
3	3 を出力
> 3	3 を入力
3	3 を出力
> 7	7 を入力
7	7 を出力
> -1	負数を入力
0 : 0	分布を出力
1 : 0	
2 : 0	
3 : 2	
4 : 0	
5 : 0	
6 : 0	
7 : 1	
9 : 0	

47

課題B5 【必修, 10点】

- アドレス from から始まる10ワードのデータを、アドレス to から始まる10ワードの領域にコピーせよ

	to	from
コピー前	0 0 0 0 0 1 2 3 4 5 6 7 8 9 10	
コピー後	1 2 3 4 5 6 7 8 9 10 6 7 8 9 10	

- 領域の重なりに注意すること
- また、正しく移動されたことを確認するために、コピー前後の15ワードの領域の内容を表示せよ
- データ領域の記述と実行結果は次ページを参考にせよ

48

データ領域の記述

```
to: .word 0,0,0,0,0  
from: .word 1,2,3,4,5  
.word 6,7,8,9,10
```

※ アドレスは本来は16進数で表示したいが、
この課題では簡単のため10進数表示でよい
※ コピー前後の境目に空行を挟むと読みやすい

空行

期待される実行結果

アドレス	値
268500992 : 0	
268500996 : 0	
268501000 : 0	
268501004 : 0	
268501008 : 0	
268501012 : 1	
268501016 : 2	
268501020 : 3	
268501024 : 4	
268501028 : 5	
268501032 : 6	
268501036 : 7	
268501040 : 8	
268501044 : 9	
268501048 : 10	

268500992 : 1
268500996 : 2
268501000 : 3
268501004 : 4
268501008 : 5
268501012 : 6
268501016 : 7
268501020 : 8
268501024 : 9
268501028 : 10
268501032 : 6
268501036 : 7
268501040 : 8
268501044 : 9
268501048 : 10

49

【補足】メモリへの書き出し

C言語

```
array[0] = 23;  
array[1] = 89;
```

アセンブリ言語

la	\$t5, array
li	\$t6, 23
SW	\$t6, 0(\$t5)
li	\$t7, 89
SW	\$t7, 4(\$t5)

- レジスタの値をメモリに転送する命令
 - SW レジスタ, 符付(レジスタ) ← word単位

課題B6 【必修, 10点】

- アドレス from から始まる10ワードのデータを、アドレス to から始まる10ワードの領域にコピーせよ

	from		to	
コピー前	1	2	3	4
コピー後	1	2	3	4

5 6 7 8 9 10 0 0 0 0

5 1 2 3 4 5 1 2 3 4 5 6 7 8 9 10

- 上の図に対応するデータ領域の記述は以下の通り

```
from: .word 1,2,3,4,5  
to: .word 6,7,8,9,10  
.word 0,0,0,0,0
```

課題B7 【選択, 5点】

- コピー対象の領域の重なり具合が、課題B5と課題B6のどちらの場合であっても、コピー方法を自動的に切り替えて、正しくコピーできるプログラムを作成せよ

- 完成したプログラムが正しく動作することを示すため、両方（課題B5, B6）のデータ領域に対する実行結果を明記すること

51

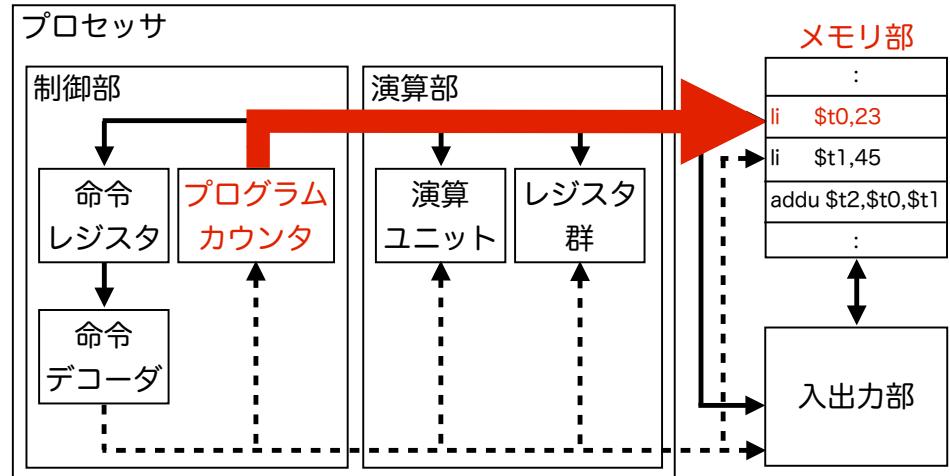
52

サブルーチン

53

【復習】 PC (Program Counter)

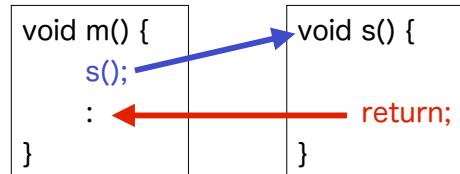
「次に実行する機械語命令の番地」を指すレジスタ



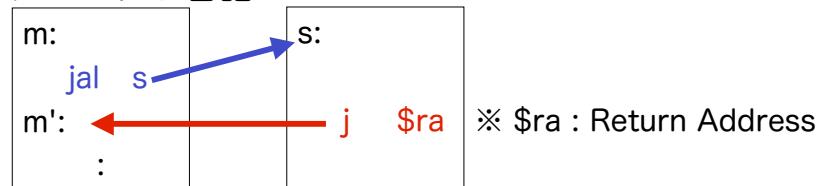
54

呼び出しと復帰

C言語



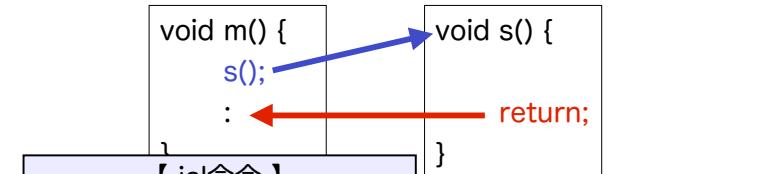
アセンブリ言語



55

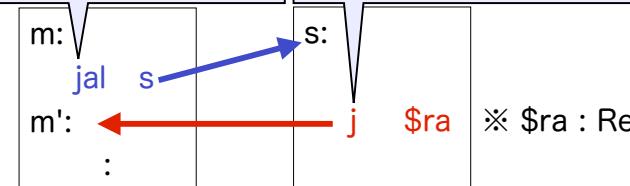
呼び出しと復帰

C言語



【jal命令】
PC(=m')を\$raに退避してから
PCに s を設定

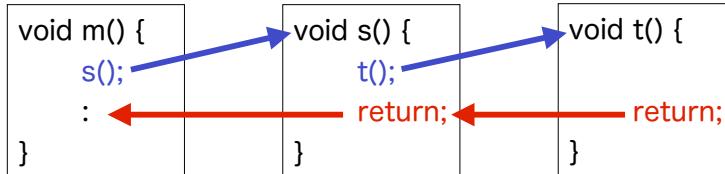
PCに \$ra (= m') を設定



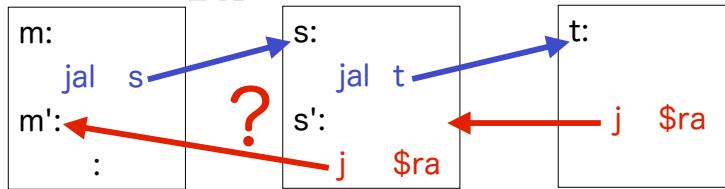
56

呼び出しと復帰

C言語



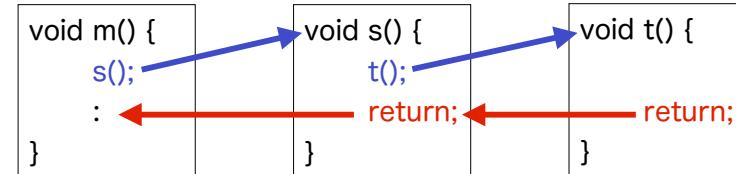
アセンブリ言語



57

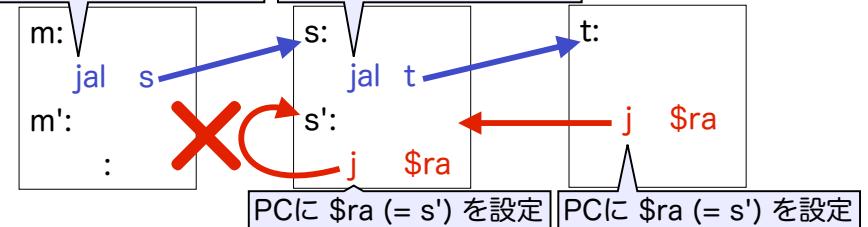
呼び出しと復帰

C言語



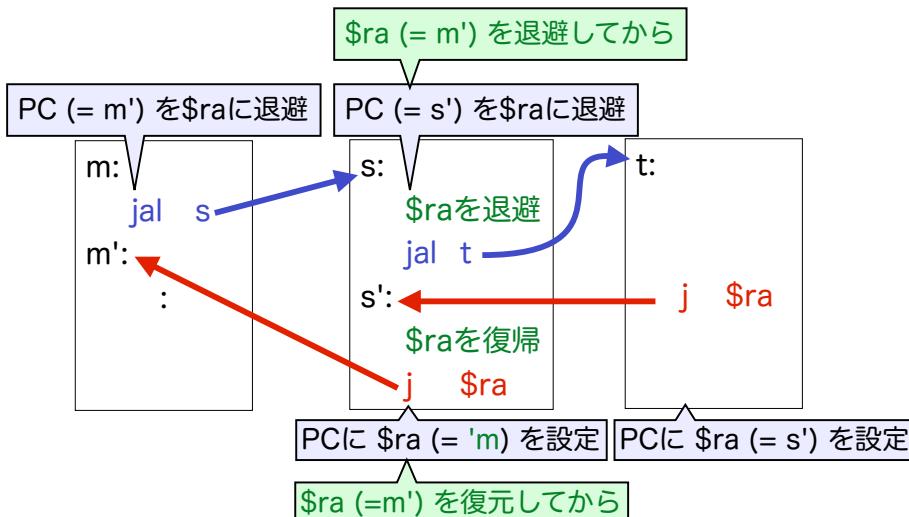
PC (= m') を \$ra に退避

PC (= s') を \$ra に退避



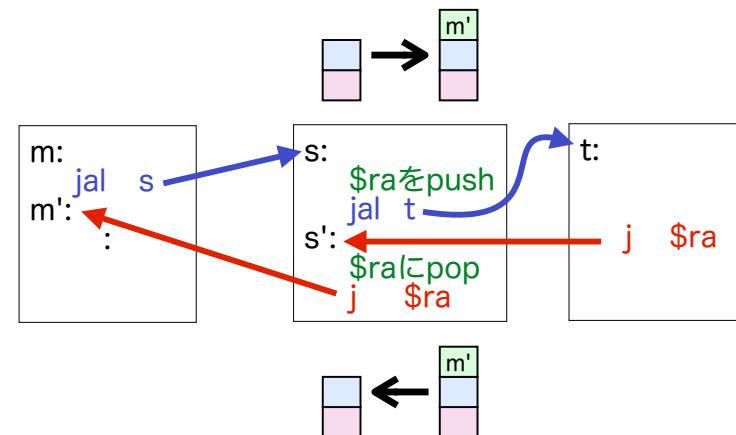
58

\$raも退避が必要



59

\$raをスタックに退避



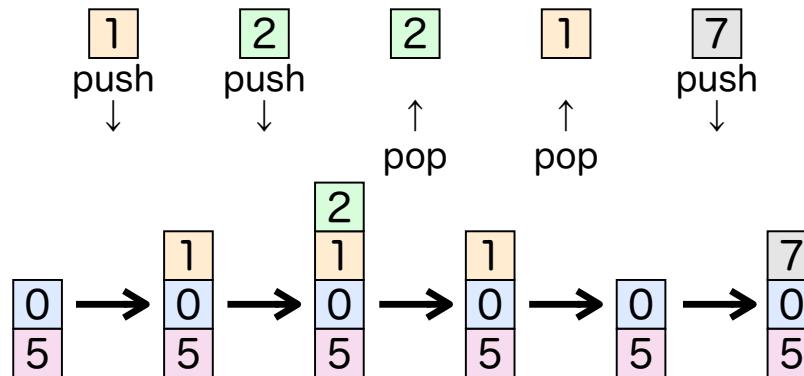
※ スタック領域はメモリ上に確保される

60

【復習】スタック

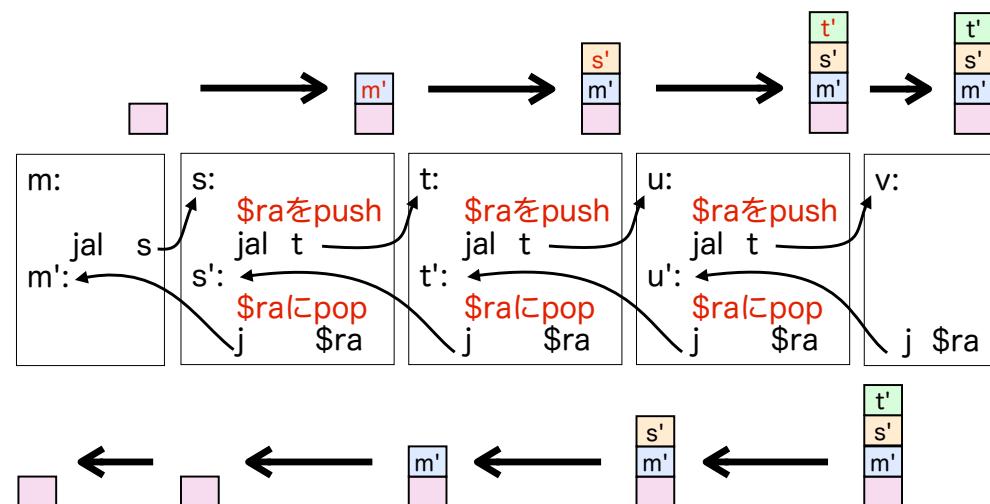
後入れ先出しのデータ構造

(LIFO : Last In First Out)



61

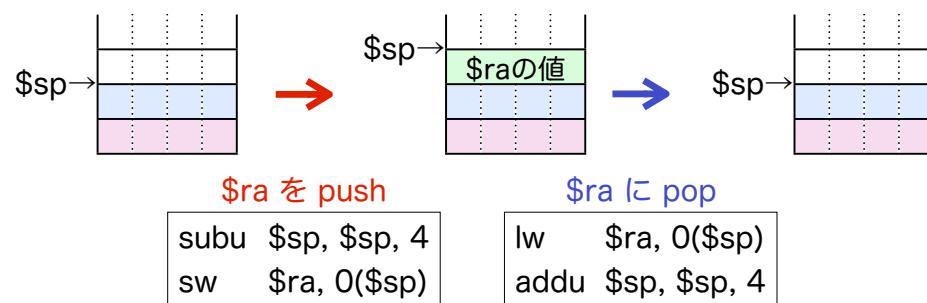
\$raをスタックに退避



62

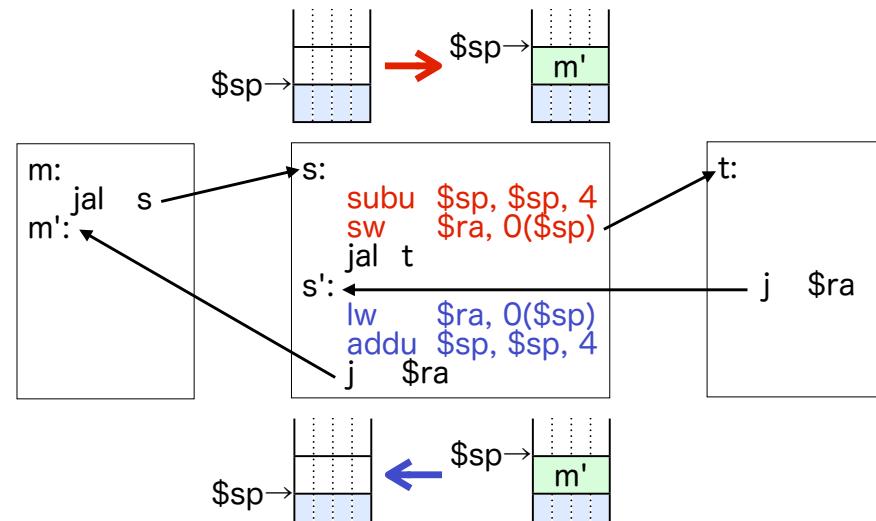
スタック操作の実現

スタックポインタ (\$sp) を用いてpushとpopを実現



63

呼び出しと復帰のまとめ



64

引数と返り値

- 引数

- 第1～4引数 → \$a0～\$a3 で授受
- 第5引数以降 → スタックで授受

- 返り値

- 返り値が1ワード → \$v0 で授受
- 返り値が2ワード → \$v0～\$v1 で授受

※ この規約により、異なる開発者間でサブルーチンを共有できる
※ また、異なる言語間でのライブラリ共有の可能性も高まる

65

引数と返り値

C言語

```
int main() {  
    int z = s(56);  
    :  
}
```

```
int s(int x) {  
    int y = x + 89;  
    return y;  
}
```

アセンブリ言語

引数を\$a0に入れてから…
main:
 li \$a0, 56
 jal s
 :
呼び出す

返り値を\$v0に入れてから…
s:
 addu \$v0, \$a0, 89
 j \$ra
 :
 復帰する

66

作業レジスタの退避

- \$t0～\$t9

- 呼び出し先で上書きされるかもしれない
→ \$t0～\$t9の値が呼び出し後も必要なら、
サブルーチンを呼び出す際に、
呼び出し元がスタックに退避し、呼び出し後に復元

- \$s0～\$s7

- 呼び出し元で既に使用されているかもしれない
→ **呼び出し先**が\$s0～\$s7を使う時は、
使用前に、レジスタの値をスタックに退避する。
使用後は、レジスタに値を復元してからreturn.

67

ローカル変数の退避

- ローカル変数はサブルーチン内でしか必要ない

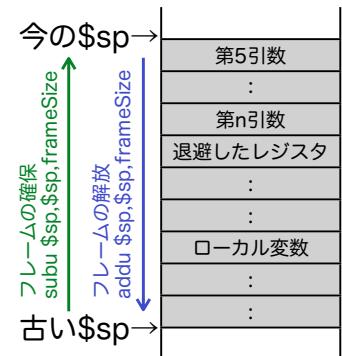
- サブルーチン開始時にスタック上に格納し、
サブルーチン終了時にスタックを畳む

- 変数の生存期間 = サブルーチンの実行期間

68

スタックフレーム

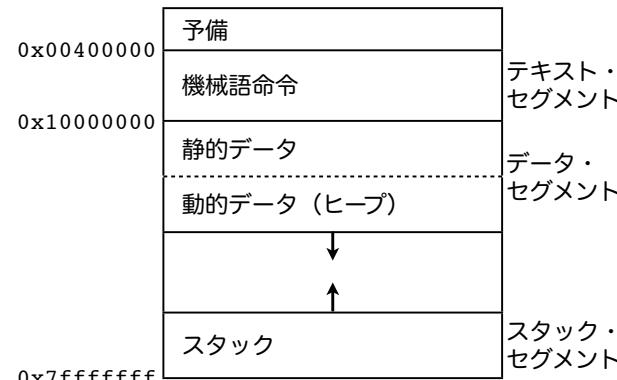
- ・スタックは、サブルーチンを呼び出す度に、伸びる
- ・1回の呼び出しで伸びる部分をスタックフレームと呼ぶ
- ・スタックフレームの内容
 - 引数 (第5引数以降)
 - 退避したレジスタの値
(\$ra, \$t0～\$t9, \$s0～\$s7)
 - ローカル変数
- ・サイズは16バイト単位に揃える



69

【補足】メモリ配置

- ・プログラムの実行開始時には、以下のようなメモリ空間が割当てられる
- ・変数やデータの置かれる場所や、スタックフレームの使われ方の理解はとても重要
- ・高級言語でプログラムを作るときでも、頭の中ではこのメモリ配置を意識する
- ・これらが分からないと、本当にソフトウェアが分かるようにはならない



70

例：フィボナッチ数

$$\text{fib}(n) = \begin{cases} 1 & (0 < n < 3) \\ \text{fib}(n-1) + \text{fib}(n-2) & (n \geq 3) \end{cases}$$

【C言語】

```
#include <stdio.h>
int main() {
    int r = fib(7);
    printf("%d", r);
    return 0;
}
```

```
int fib(int n) {
    int x, y, w;
    if (n < 3) {
        w = 1;
    } else {
        x = fib(n-1);
        y = fib(n-2);
        w = x + y;
    }
    return w;
}
```

フィボナッチ数
【アセンブリ言語】

メインルーチン

```
.text
.align 2
.globl main
main:
    subu    $sp,$sp,16    # フレームの確保
    sw     $ra,0($sp)    # 戻り番地の退避
    li     $a0,7          # r = fib(7)
    jal     fib
    move   $a0,$v0        # printf("%d",r)
    li     $v0,1
    syscall
    lw     $ra,0($sp)    # 戻り番地の復元
    addu   $sp,$sp,16    # フレームの解放
    move   $2,$0          # return 0
    j      $ra
```

71

72

フィボナッチ数

【アセンブリ言語】

サブルーチン fib

```
fib:    subu $sp,$sp,16 # フレームの確保
        sw    $ra,0($sp) # 戻り番地の退避
        slt   $t0,$a0,3  # if (n < 3) ...
        beq   $t0,$0,els
        li    $v0,1      # w = 1
        j     ret

els:    sw    $s0,4($sp) # $s0の退避
        sw    $s1,8($sp) # $s1の退避
        move  $s0,$a0
        subu $a0,$a0,1  # x = fib(n-1)
        jal   fib
        move  $s1,$v0
        subu $a0,$s0,2  # y = fib(n-2)
        jal   fib
        addu $v0,$s1,$v0 # w = x + y
        lw    $s0,4($sp) # $s0の復元
        lw    $s1,8($sp) # $s1の復元
ret:    lw    $ra,0($sp) # 戻り番地の復元
        addu $sp,$sp,16 # フレームの解放
        j     $ra         # return w
```

73

【補足】 乗算命令

- 乗算「レジスタA × レジスタB」を行う命令
 - mult レジスタA, レジスタB ← 符号つき乗算
 - multu レジスタA, レジスタB ← 符号なし乗算
- 乗算結果（64ビット）は特殊なレジスタ Hi と Lo に入る
 - 上位32ビットは Hi
 - 下位32ビットは Lo
- レジスタ Hi や Lo の値を、普通のレジスタに移すには、以下の特殊な転送命令を用いる
 - mfhi レジスタA ← Hiの値をレジスタAに転送
 - mflo レジスタA ← Loの値をレジスタAに転送

課題B8 【必修, 10点】

- 階乗を再帰的に求めるサブルーチン fact を作成せよ

$$\text{fact}(n) = \begin{cases} 1 & (n = 0) \\ n \times \text{fact}(n-1) & (n > 0) \end{cases}$$

- まず、フィボナッチ数の例を参考にしてC言語で作成せよ
- 次に、そのC言語版を基に、アセンブリ言語で作成せよ
- 作成したサブルーチン fact を用いて、10から0までの階乗の数表を右の図のように表示するプログラムを作成せよ

10! = 3628800
9! = 362880
8! = 40320
:
0! = 1

74

課題B9 【選択, 5点】

- 階乗を（再帰ではなく）繰り返しによって求めるサブルーチンを作成せよ
- 課題B8と同様に、作成したサブルーチンを用いて10から0までの階乗の数表を表示するプログラムを作成せよ

75

76

応用問題

77

手順

- 筆算の手順を模倣する
- 手で行う筆算 … 1桁毎に掛け算を行って繰り上げ処理を行う
- すこし効率良くするために、4桁毎に区切って計算する
 - 4桁の10進数を1桁として表現（10000進数）
- 10000進数の桁ごとに掛け算と繰り上げの処理を行う

$$\begin{array}{r} 62 & 2702 & 0800 \\ \times & & 14 \text{ 掛け算} \\ \hline 868 & 37828 & 11200 \\ + & 3 \leftarrow & 1 \leftarrow \text{ 繰り上げ} \\ \hline 871 & 7829 & 1200 \end{array}$$

- 100の階乗の桁数は10進158桁（10000進40桁）
→ 計算に必要なメモリ領域は、32ビット整数 × 40個分

課題B10 【選択、5点】

- 100の階乗、すなわち、

```
100! = 0093 3262 1544 3944 1526 8169 9238 8562 6670 0490  
7159 6826 4381 6214 6859 2963 8952 1759 9993 2299  
1560 8941 4639 7615 6518 2862 5369 7920 8272 2375  
8251 1852 1091 6864 0000 0000 0000 0000 0000 0000
```

を計算し、表示するプログラムを作成せよ

- 次ページの手順、および、
次々ページのC言語のプログラムを参考にせよ

78

【C言語】

```
#include <stdio.h>  
unsigned int result[40];  
int main() {  
    unsigned int top = 0; // 0 でない最も上の桁 (天井)  
    unsigned int kake; // 乗数 (a × b の b)  
    unsigned int agari; // 繰り上がり  
    int i, t;  
    result[0] = 1;  
    for (kake = 2; kake <= 100; kake++) {  
        agari = 0;  
        for (i = 0; i <= top; i++) {  
            t = result[i] * kake; // 下の桁から1桁ずつ掛けていく  
            t = t + agari; // 下からの上がりを足しこむ  
            agari = t / 10000; // 上の桁への上がり  
            result[i] = t % 10000; // 余りはこの桁に残す  
        }  
        if (agari > 0) { // 今の天井を超えたたら…  
            top++; // 天井を更に押し上げる  
            result[top] = agari; // 繰り上がりを乗せる  
        }  
    }  
    for (i = top; i >= 0; i--) // 上の桁 (天井) から表示していく  
        printf("%d", result[i]);  
    return 0;  
}
```

79

80

【補足】除算命令

- 除算「レジスタA / レジスタB」を行う命令
 - div レジスタA, レジスタB ← 符号つき除算
 - divu レジスタA, レジスタB ← 符号なし除算
- 商と剰余は、レジスタ Lo と Hi に入る
 - 商は Lo
 - 剰余は Hi

81

【補足】.word 指令

初期化されたデータ領域の静的確保

```
.data
a: .word 0:16 # 初期値 0 のワード x 16個
b: .word 7:3 # 初期値 7 のワード x 3個
```

82

【補足】表示の崩れ

int 表示用の（番号1の）システムコールを用いただけでは、プログラムの表示結果は、以下のように、崩れてしまう

```
93326215443944152681699238856266704907
1596826438162146859296389521759999322991
5608941463976156518286253697920827223758
251185210916864000000
```

各桁の表示の直後に空白を表示するように（わずか4行程度の追加を）するだけでも、格段に読みやすくなる

```
93 3262 1544 3944 1526 8169 9238 8562 6670 490
7159 6826 4381 6214 6859 2963 8952 1759 9993 2299
1560 8941 4639 7615 6518 2862 5369 7920 8272 2375
8251 1852 1091 6864 0 0 0 0 0
```

83

課題B11 【選択、5点】

- 実行結果が以下のように正しく表示されるように、課題B10で作成したプログラムを改良せよ

```
0093 3262 1544 3944 1526 8169 9238 8562 6670 0490
7159 6826 4381 6214 6859 2963 8952 1759 9993 2299
1560 8941 4639 7615 6518 2862 5369 7920 8272 2375
8251 1852 1091 6864 0000 0000 0000 0000 0000 0000
```

- 崩れの解消には10000進数の各桁を表示する直前に、以下のような処理を追加すればよい

- 1000未満なら'0'をひとつ表示
- 100未満なら'0'をひとつ表示
- 10未満なら'0'をひとつ表示

- また、1行の表示桁数を10個（または8個）ずつに揃えよ

84

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)	OPCODE/FUNCT (Hex)
Add	add	R	$R[rd]=R[rs]+R[rt]$	(1) 0/20
Add Immediate	addi	I	$R[rt]=R[rs]+SignExtImm$	(1)(2) 8
Add Imm. Unsigned	addiu	I	$R[rt]=R[rs]+SignExtImm$	(2) 9
Add Unsigned	addu	R	$R[rd]=R[rs]+R[rt]$	(2) 0/21
Subtract	sub	R	$R[rd]=R[rs]-R[rt]$	(1) 0/22
Subtract Unsigned	subu	R	$R[rd]=R[rs]-R[rt]$	(0/23)
And	and	R	$R[rd]=R[rs]\&R[rt]$	0/24
And Immediate	andi	I	$R[rt]=R[rs]\&ZeroExtImm$	(3) c
Nor	nor	R	$R[rd]=\sim(R[rs]\ R[rt])$	0/27
Or	or	R	$R[rd]=R[rs]\ R[rt]$	0/25
Or Immediate	ori	I	$R[rt]=R[rs]\ ZeroExtImm$	(3) d
Xor	xor	R	$R[rd]=R[rs]\text{`}R[rt]$	0/26
Xor Immediate	xori	I	$R[rt]=R[rs]\text{`}ZeroExtImm$	e
Shift Left Logical	sll	R	$R[rd]=R[rs]\ll shamt$	0/00
Shift Right Logical	srl	R	$R[rd]=R[rs]\gg shamt$	0/02
Shift Right Arithmetic	sra	R	$R[rd]=R[rs]\gg> shamt$	0/03
Shift Left Logical Var.	sllv	R	$R[rd]=R[rs]\ll R[rt]$	0/04
Shift Right Logical Var.	srlv	R	$R[rd]=R[rs]\gg R[rt]$	0/06
Shift Right Arithmetic Var.	sraev	R	$R[rd]=R[rs]\gg> R[rt]$	0/07
Set Less Than	slt	R	$R[rd]=(R[rs]<R[rt])?1:0$	0/2a
Set Less Than Imm.	slti	I	$R[rt]=(R[rs]<SignExtImm)?1:0$	(2) a
Set Less Than Imm. Unsigned	sltiu	I	$R[rt]=(R[rs]<SignExtImm)?1:0$	(2)(6) b
Set Less Than Unsigned	sltu	R	$R[rd]=(R[rs]<R[rt])?1:0$	(6) 0/2b
Branch On Equal	beq	I	if($R[rs]==R[rt]$) PC=PC+4+BranchAddr	(4) 4
Branch On Not Equal	bne	I	if($R[rs]!=R[rt]$) PC=PC+4+BranchAddr	(4) 5
Branch Less Than	blt	P	if($R[rs]<R[rt]$) PC=PC+4+BranchAddr	
Branch Greater Than	bgt	P	if($R[rs]>R[rt]$) PC=PC+4+BranchAddr	
Branch Less Than Or Equal	ble	P	if($R[rs]\leq R[rt]$) PC=PC+4+BranchAddr	
Branch Greater Than Or Equal	bge	P	if($R[rs]\geq R[rt]$) PC=PC+4+BranchAddr	
Jump	j	J	PC=JumpAddr	(5) 2
Jump And Link	jal	J	$R[31]=PC+4;$ PC=JumpAddr	(5) 2
Jump Register	jr	R	PC=R[rs]	0/08
Jump And Link Register	jalr	R	$R[31]=PC+4;$ PC=R[rs]	0/09
Move	move	P	$R[rd]=R[rs]$	
Load Byte	lb	I	$R[rt]=\{24'b0, M[R[rs]+ZeroExtImm](7:0)\}$	(3) 20
Load Byte Unsigned	lbu	I	$R[rt]=\{24'b0, M[R[rs]+SignExtImm](7:0)\}$	(2) 24
Load Halfword	lh	I	$R[rt]=\{16'b0, M[R[rs]+ZeroExtImm](15:0)\}$	(3) 25
Load Halfword Unsigned	lhu	I	$R[rt]=\{16'b0, M[R[rs]+SignExtImm](15:0)\}$	(2) 25
Load Upper Imm.	lui	I	$R[rt]=\{imm, 16'b0\}$	f
Load Word	lw	I	$R[rt]=M[R[rs]+SignExtImm]$	(2) 23
Load Immediate	li	P	$R[rd]=\text{immediate}$	
Load Address	la	P	$R[rd]=\text{immediate}$	
Store Byte	sb	I	$M[R[rs]+SignExtImm](7:0)=R[rt](7:0)$	(2) 28
Store Halfword	sh	I	$M[R[rs]+SignExtImm](15:0)=R[rt](15:0)$	(2) 29
Store Word	sw	I	$M[R[rs]+SignExtImm]=R[rt]$	(2) 2b

REGISTERS

NAME	NMBR	USE	STORE?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes
\$f0-\$f31	0-31	Floating Point Registers	Yes

- (1) May cause overflow exception
- (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
- (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
- (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
- (4) $\text{JumpAddr} = \{\text{PC}[31:28], \text{address}, 2'b0\}$
- (6) Operands considered unsigned numbers (vs. 2's comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS

R	$^{31}_{31}$	$^{opcode}_{26/25}$	$^{rs}_{21/20}$	$^{rt}_{16/15}$	$^{rd}_{11/10}$	$^{shamt}_{6/5}$	$^{funct}_{0}$
I	$^{31}_{31}$	$^{opcode}_{26/25}$	$^{rs}_{21/20}$	$^{rt}_{16/15}$			$^{immediate}_{0}$
J	$^{31}_{31}$	$^{opcode}_{26/25}$					$^{immediate}_{0}$
FR	$^{31}_{31}$	$^{opcode}_{26/25}$	$^{fmt}_{21/20}$	$^{ft}_{16/15}$	$^{fs}_{11/10}$	$^{fd}_{6/5}$	$^{funct}_{0}$
FI	$^{31}_{31}$	$^{opcode}_{26/25}$	$^{fmt}_{21/20}$	$^{rt}_{16/15}$			$^{immediate}_{0}$

ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)	OPCODE/FMT/FT/FUNCT
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/-/-/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6) 0/-/-/1b
Multiply	mult	R	{Hi,Lo}=R[rs]*R[rt]	0/-/-/18
Multiply Unsigned	multu	R	{Hi,Lo}=R[rs]*R[rt]	(6) 0/-/-/19
Branch On FP True	bclt	FI	if(FPCond) PC=PC+4+BranchAddr	(4) 11/8/1/-
Branch On FP False	bclf	FR	if(!FPCond) PC=PC+4+BranchAddr	(4) 11/8/0/-
FP Compare Single	c.x.s*	FR	FPCond=(F[fs] op F[ft])?1:0	11/10/-/y
FP Compare Double	c.x.d*	FR	FPCond={({F[fs],F[fs+1]} op {F[ft],F[ft+1]})?1:0 *(x is eq, lt or le) (op is ==, < or <=) (y is 32, 3c or 3e)}	11/11/-/y
FP Add Single	add.s	FR	F[fd]=F[fs]+F[ft]	11/10/-/0
FP Divide Single	div.s	FR	F[fd]=F[fs]/F[ft]	11/10/-/3
FP Multiply Single	mul.s	FR	F[fd]=F[fs]*F[ft]	11/10/-/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs]-F[ft]	11/10/-/1
FP Add Double	add.d	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}+{F[ft],F[ft+1]}	11/11/-/0
FP Divide Double	div.d	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}/{F[ft],F[ft+1]}	11/11/-/3
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}*{F[ft],F[ft+1]}	11/11/-/2
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}	11/11/-/1
Move From Hi	mfhi	R	R[rd]=Hi	0/-/-/10
Move From Lo	mflo	R	R[rd]=Lo	0/-/-/12
Move From Control	mfc0	R	R[rd]=CR[rs]	16/0/-/0
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2) 31/-/-/-
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/-/-/-
Store FP Single	swc1	I	M[R[rs]+SignExtImm]=F[rt]	(2) 39/-/-/-
Store FP Double	sdc1	I	M[R[rs]+SignExtImm]=F[rt]; M[R[rs]+SignExtImm+4]=F[rt+1]	(2) 3d/-/-/-

ASSEMBLER DIRECTIVES

.data [addr]*	Subsequent items are stored in the data segment
.kdata [addr]*	Subsequent items are stored in the kernel data segment
.ktext [addr]*	Subsequent items are stored in the kernel text segment
.text [addr]*	Subsequent items are stored in the text * starting at [addr] if specified
.ascii str	Store string str in memory, but do not null-terminate it
.asciiz str	Store string str in memory and null-terminate it
.byte b ₁ ,...,b _n	Store the n values in successive bytes of memory
.double d ₁ ,...,d _n	Store the n floating-point double precision numbers in successive memory locations
.float f ₁ ,...,f _n	Store the n floating-point single precision numbers in successive memory locations
.half h ₁ ,...,h _n	Store the n 16-bit quantities in successive memory halfwords
.word w ₁ ,...,w _n	Store the n 32-bit quantities in successive memory words
.space n	Allocate n bytes of space in the current segment
.extern symsize	Declare that the datum stored at sym is size bytes large and is a global label
.globl sym	Declare that label sym is global and can be referenced from other files
.align n	Align the next datum on a 2 ⁿ byte boundary, until the next .data or .kdata directive
.set at	Tells SPIM to complain if subsequent instructions use \$at
.set noat	prevents SPIM from complaining if subsequent instructions use \$at

SYSCALLS

SERVICE	\$v0	ARGS	RESULT
print_int	1	integer \$a0	
print_float	2	float \$f12	
print_double	3	double \$f12/\$f13	
print_string	4	string \$a0	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	buf \$a0, buflen \$a1	
sbrk	9	amount \$a	address (in \$v0)
exit	10		

EXCEPTION CODES

Number	Name	Cause of Exception
0	Int	Interrupt (hardware)
4	AdEL	Address Error Exception (load or instruction fetch)
5	AdES	Address Error Exception (store)
6	IBE	Bus Error on Instruction Fetch
7	DBE	Bus Error on Load or Store
8	Sys	Syscall Exception
9	Bp	Breakpoint Exception
10	RI	Reserved Instruction Exception
11	CpU	Coprocessor Unimplemented
12	Ov	Arithmetic Overflow Exception
13	Tr	Trap
15	FPE	Floating Point Exception

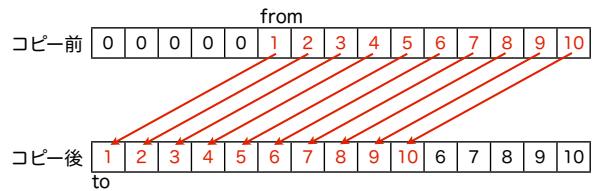
[1] Patterson, David A; Hennessy, John J.: Computer Organization and Design, 3rd Edition. Morgan Kaufmann Publishers. San Francisco, 2005.

計算機工学実験I

課題B5とB6のヒント

森本

課題B5【必修, 10点】



1

2

データ領域の記述

```
to: .word 0,0,0,0,0
from: .word 1,2,3,4,5
        .word 6,7,8,9,10
```

※ アドレスは本来は16進数で表示したいが、
この課題では簡単のため10進数表示でよい
※ コピー前後の境目に空行を挟むと読みやすい

期待される実行結果

アドレス	値
268500992 : 0	268500992 : 1
268500996 : 0	268500996 : 2
268501000 : 0	268501000 : 3
268501004 : 0	268501004 : 4
268501008 : 0	268501008 : 5
268501012 : 1	268501012 : 6
268501016 : 2	268501016 : 7
268501020 : 3	268501020 : 8
268501024 : 4	268501024 : 9
268501028 : 5	268501028 : 10
268501032 : 6	268501032 : 6
268501036 : 7	268501036 : 7
268501040 : 8	268501040 : 8
268501044 : 9	268501044 : 9
268501048 : 10	268501048 : 10

空行

3

まずはC言語で考える

データ領域を確保

```
int main() {
    // コピー前の領域の内容を表示
    // データをコピー
    // 空行を表示
    // コピー後の領域の内容を表示
    return 0;
}
```

4

データ領域の確保

```
array to from
      0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10
```

アセンブリ言語

```
array:
to: .word 0,0,0,0,0
from: .word 1,2,3,4,5
        .word 6,7,8,9,10
```

C言語

```
int array[] = {0,0,0,0,0,
                1,2,3,4,5,
                6,7,8,9,10};
int *to    = array;
int *from = array + 5;
```

5

データ領域の表示

```
array 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10
```

↑
a

```
int i;
int *a = array;
for (i = 15; i != 0; i--) {
    printf("%d : %d\n", a, *a);
    a++;
}
```

アドレス

そのアドレスに
格納された値

6

データ領域の表示

```
array 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10
```

↑
a

```
char *colon = ":";  
char *newline = "\n";  
int i = 15;  
int *a = array;  
do {  
    printf("%d", a);  
    printf(colon);  
    printf("%d", *a);  
    printf(newline);  
    a++;  
    i--;  
} while (i != 0);
```

・実行内容は
前ページと同じ
・アセンブリ言語
に近い記述

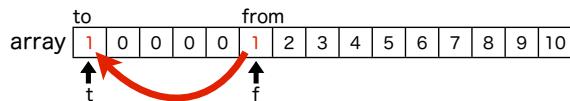
データ領域を表示する アセンブリプログラム

```
.data
colon: .asciiz ":"
newline: .asciiz "\n"
array: .word 0,0,...  
.
.text
loop:
    li    $t0, 15      # int i = 15
    la    $t1, array   # int *a = array
    subu $t0, $t0, 1    # do {
    syscall                   # printf("%d", a)
    li    $v0, 1          # printf(colon)
    la    $a0, colon      # printf(":")
    syscall                   # printf("%d", *a)
    li    $v0, 1          # printf("%d", *a)
    lw    $a0, 0($t1)     # printf("\n")
    syscall                   # printf("\n")
    addu $t1, $t1, 4      # a ++
    subu $t0, $t0, 1      # i --
    bne  $t0, $0, loop    # } while (i != 0)
    :
```

7

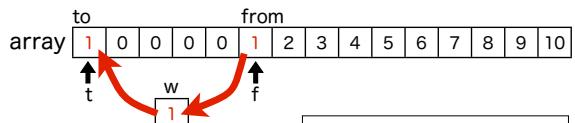
8

データのコピー



```
int i;
int *f = from;
int *t = to;
for (i = 10; i != 0; i--) {
    *t = *f;
    f++;
    t++;
}
```

データのコピー



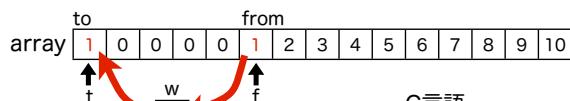
```
int w;
int i = 10;
int *f = from;
int *t = to;
do {
    w = *f;
    *t = w;
    f++;
    t++;
    i--;
} while (i != 0);
```

- 実行内容は前ページと同じ
- アセンブリ言語に近い記述

9

10

データのコピー



C言語

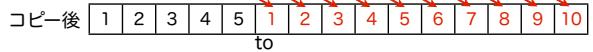
```
la    $t2, from .....  
la    $t3, to.....  
  
lw    $t4, 0($t2).....  
sw    $t4, 0($t3).....  
  
同等のアセンブリ命令
```

```
int w;
int i = 10;
int *f = from;
int *t = to;
do {
    w = *f;
    *t = w;
    f++;
    t++;
    i--;
} while (i != 0);
```

11

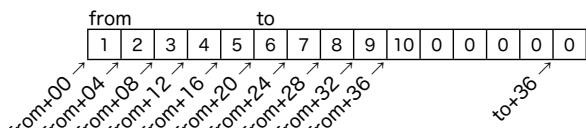
12

課題B6 【必修, 10点】



from の先頭からコピーを始めると、まだコピーを終えていない値を上書きしてしまうことに注意。例えば、from の先頭の「1」を to の先頭にコピーすると「6」は上書きされてしまう。ゆえに、「6」が上書きされる前に、「6」のコピーを済ませておく必要がある。

【補足】 データ領域の番地



la 命令のオペランドには、
以下の単純な定数式も記述できる

```
la $t3, to+36
```

【お願い】 字下げを揃えましょう

ラベル	命令	オペランド
str:	.data	"Hello, World\n"
	.asciiz	
main	.text	2
	.align	main
	.globl	
main	li	\$v0,4
	la	\$a0,str
	syscall	
	move	\$v0,\$0
	j	\$ra

13

14