CSL
Computer Systems Laboratory

情報科学演習
プロセッサ演習
～第４週～

情報科学科
コンピュータシステム研究室

プログラミング言語

- 機械語（machine lang.）
  ➔2進数の言語、直接理解するのは困難
- アセンブリ言語（assembly lang.）
  ➔記号形式の言語、1つの命令が1行の記号と対応
- 高水準プログラミング言語（high-level programming Lang.）
  ➔英語と代数式による言語、プログラマは理解しやすい

機械語
（MIPS）

```
00000010 00110010 01000000 00100000
```

アセンブラ
（assembler）

アセンブリ言語
（MIPS）

```
add t0, s1, s2
```

コンパイラ
（compiler）

高水準プログラミング言語
（C）

```
t0 = s1 + s2;
```

GCC-MIPSコンパイラの使い方

■ インストール

- Ubuntu:「sudo apt install gcc-mips-linux-gnu」
- Windows: 以下のいずれかの環境でUbuntuを起動
  - WSL (Windows Subsystem for Linux)
  - 仮想マシン (Virtural PC, Virtual Box, Vmwareなど)

■実行

- mips-linux-gnu-gcc –S file.c
  - –fverbose-asm を追加：Cソースの記述も表示
  - –O1 または –O2 を追加：コードを最適化

GCC-MIPSコンパイラの使い方

■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

例：C言語

clear_array.c

```
void clear_array(int* x, int n)
{
  int i, j, tmp;
  i = 0;
  do{
    x[i] = 0;
    i++;
  } while(i <= n);

  return;
}
```

```
int main(void)
{
  int array[11] ={5,8,…,4,6};
  int i;

  show_array(array, 10);
  clear_array(array, 0, 10);
  show_array(array, 10);

  return 0;
}
```

ソートのプログラムは
関数で記述する

main関数から呼び出す

例：アセンブリ言語

```
mips-linux-gnu-gcc -S -O1 clear_array.c
```

遅延スロットあり

clear_array.s

```
clear_array:
  .frame  $sp,0,$31
  .mask   0x00000000,0
  .fmask  0x00000000,0
  .set    noreorder
  .set    nomacro
  move    $2,$0       # i=0;
$L2:                  # do{
  sw      $0,0($4)    # x[i]=0;
  addiu   $2,$2,1     # i++;
  slt     $3,$5,$2    # (n<i?)
  beq     $3,$0,$L2   # }while(i<=n);
  addiu   $4,$4,4     # n++; (delay)

  jr      $31         # return;
  nop
```

```
void clear_array(int* x, int n){
    ...
    return;
}
```

Name	Number
$zero	0
$at	1
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$k0-$k1	26-27
$gp	28
$sp	29
$fp	30
$ra	31

引数

$4 = $a0
配列先頭番地

$5 = $a1
配列末尾番号

戻りアドレス

$31 = $ra
呼出し元位置

例：アセンブリ言語

該当部分をコピー＆ペースト
引数を渡して関数呼び出し

clear_array.asm

```
#####################
#↓以下に並び替えの...
addu    $4, $zero, $s0
addiu   $5, $zero, 99
jal     clear_array
nop
j       Done
nop

clear_array:
    ...
    jr      $31
    nop
    # ↑ここまで
#####################
Done:
```

```
void clear_array(int* x, int n){
    ...
    return;
}
```

Name	Number
$zero	0
$at	1
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$k0-$k1	26-27
$gp	28
$sp	29
$fp	30
$ra	31

引数

$4 = $a0
配列先頭番地

$5 = $a1
配列末尾番号

戻りアドレス

$31 = $ra
呼出し元位置

手続きのコールとリターン

```
int main(void)
{
  int f, g, h, i, j;
  ...
  f = proc(g, h, i, j);
  ...
  return 0;
}
```

```
int proc(int g, int h, int i, int j)
{
  int f;
  f = (g + h) - (i + j);
  return f;
}
```

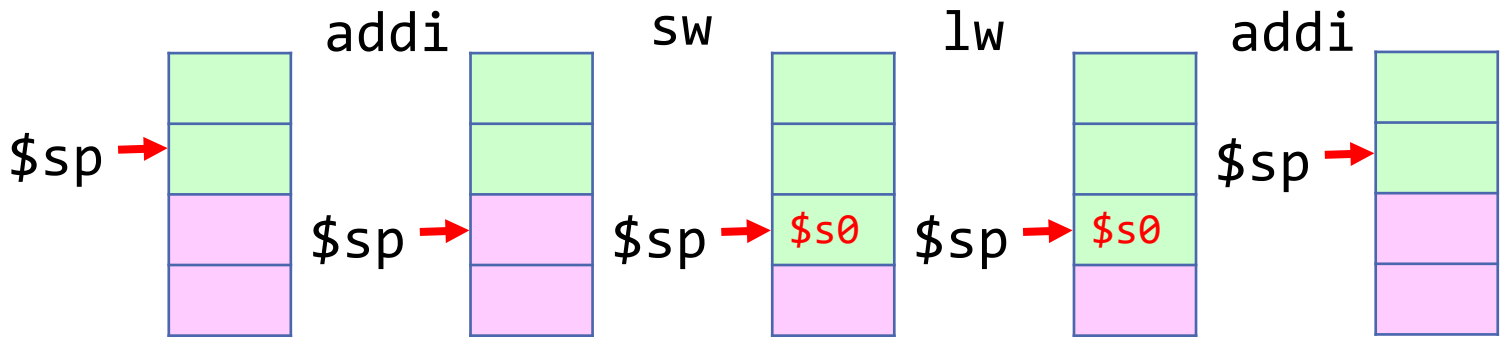
```
main: ...
      addi a0,s0,0
      addi a1,s1,0
      addi a2,s2,0
      addi a3,s3,0
      jal  proc
      addi s4,v0,0
      ...
```

```
proc: add $t0, $a0, $a1
      add $t1, $a2, $a3
      sub $s0, $t0, $t1
      add $v0, $s0, $zero
      jr  $ra
```

スタックへの退避と復元

```
proc:
        add $t0, $a0, $a1
        add $t1, $a2, $a3
        sub $s0, $t0, $t1
        add $v0, $s0, $zero
        jr  $ra
```

```
proc:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)
        add  $t0, $a0, $a1
        add  $t1, $a2, $a3
        sub  $s0, $t0, $t1
        add  $v0, $s0, $zero
        lw   $s0, 0($sp)
        addi $sp, $sp, 4
        jr  $ra
```



再帰とスタック

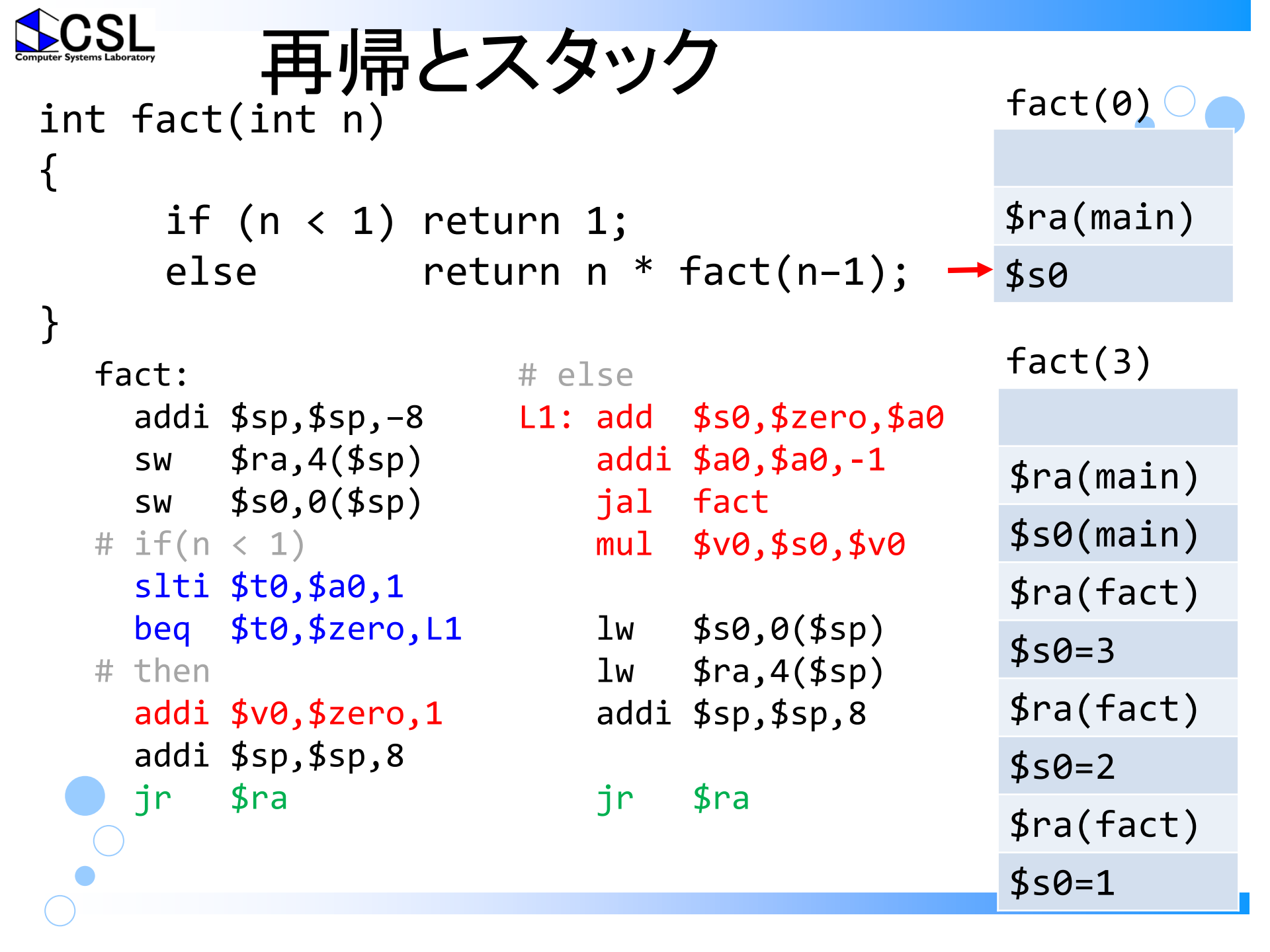
```
int fact(int n)
{
      if (n < 1) return 1;
      else       return n * fact(n-1);
}
```

```
   fact:
     addi $sp,$sp,-8
     sw   $ra,4($sp)
     sw   $s0,0($sp)
   # if(n < 1)
     slti $t0,$a0,1
     beq  $t0,$zero,L1
   # then
     addi $v0,$zero,1
     addi $sp,$sp,8
     jr   $ra
```

```
   # else
   L1: add  $s0,$zero,$a0
       addi $a0,$a0,-1
       jal  fact
       mul  $v0,$s0,$v0

       lw   $s0,0($sp)
       lw   $ra,4($sp)
       addi $sp,$sp,8

       jr   $ra
```



注意事項

以下の記述はそのままでもほとんど問題なし

```
    .frame  $sp,0,$31
    .mask   0x00000000,0
    .fmask  0x00000000,0
    .set    noreorder
    .set    nomacro
```

エラーが出る記述は削除
(上記をすべて削除してもOK)

Verilog版MIPSのメモリの終端：0x10017ffc

```
    li      $sp,0x10017ffc
```

スタックポインタを初期化

Verilog版MIPSで未定義の命令が出力される場合

PC = LABEL if $9 <= 0

```
    blez    $9, $L1
```

定義済み命令で置き換え

```
    slt     $8, $0, $9
    beq     $8, $0, $L1
```

PC = LABEL

```
    b       $L1
```

定義済み命令で置き換え

```
    j       $L1
```

Marsでアセンブル後に
定義済み命令に置換される
疑似命令は問題なし

第４週の課題

- clear_array.c: 配列初期化のCプログラム
  - Marsで動作確認
    - 100個の配列が0に初期化されているか？
  - iverilogとFPGAで動作確認
    - Marsと同じ結果になるか？実行クロック数は？
- ???_sort.c: ソートのCプログラム
  - gcc でCプログラムの動作確認
  - シミュレーションと実装による動作確認
    - 最適化オプション –O1 と –O2 で結果の違いは？
    - 実行クロック数は？

クイックソート（ピボットで分類）

ピボットより　大きいデータ：左へ
小さいデータ：右へ

4	2	0	7	3	1	6	5	8	9
4	9	0	7	3	1	6	5	8	2
4	9	8	7	3	1	6	5	0	2
4	9	8	7	5	1	6	3	0	2
4	9	8	7	5	6	1	3	0	2
6	9	8	7	5	4	1	3	0	2

6	9	8	7	5
7	9	8	6	5

1	3	0	2	
1	3	2	0	
2	3	1	0	

7	9	8
8	9	7

2	3	
3	2	

8	9	
9	8	

命令形式

算術演算命令
ロード／ストア命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	add	0	入力1	入力2	出力	0	32
	addu	0	入力1	入力2	出力	0	33
	sub	0	入力1	入力2	出力	0	34
	subu	0	入力1	入力2	出力	0	35
I	addi	8	入力	出力	即値（符号付き）		
	addiu	9	入力	出力	即値（符号なし）		
	lw	35	入力	出力	アドレス		
	sw	43	入力	出力	アドレス		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

論理演算命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	and	0	入力1	入力2	出力	0	36
	or	0	入力1	入力2	出力	0	37
	xor	0	入力1	入力2	出力	0	38
	nor	0	入力1	入力2	出力	0	39
I	andi	12	入力	出力	即値（符号なし）		
	ori	13	入力	出力	即値（符号なし）		
	xori	14	入力	出力	即値（符号なし）		
	lui	15	0	出力	即値（符号なし）		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

シフト演算命令
比較演算命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	sll	0	0	入力	出力	シフト量	0
	srl	0	0	入力	出力	シフト量	2
	sra	0	0	入力	出力	シフト量	3

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	slt	0	入力1	入力2	出力	0	42
	sltu	0	入力1	入力2	出力	0	43
I	slti	10	入力	出力	即値（符号付き）		
	sltiu	11	入力	出力	即値（符号なし）		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

条件分岐／ジャンプ命令
コール／リターン命令

形式	命令	op	rs	rt	address/immediate
I	フィールド長	6	5	5	16
	beq	4	入力1	入力2	分岐先
	bne	5	入力1	入力2	分岐先
J	j	2	ジャンプ先		
	jal	3	ジャンプ先		
	フィールド長	6	26		
形式	命令	op	target address		

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	jr	0	入力	0	0	0	8
J	jal	3	ジャンプ先				
	フィールド長	6	26				
形式	命令	op	target address				

命令一覧(形式別)

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	sll	0	0	入力	出力	シフト量	0
	srl	0	0	入力	出力	シフト量	2
	sra	0	0	入力	出力	シフト量	3
	jr	0	入力	0	0	0	8
	add	0	入力1	入力2	出力	0	32
	addu	0	入力1	入力2	出力	0	33
	sub	0	入力1	入力2	出力	0	34
	subu	0	入力1	入力2	出力	0	35
	and	0	入力1	入力2	出力	0	36
	or	0	入力1	入力2	出力	0	37
	xor	0	入力1	入力2	出力	0	38
	nor	0	入力1	入力2	出力	0	39
	slt	0	入力1	入力2	出力	0	42
	sltu	0	入力1	入力2	出力	0	43

CSL
Computer Systems Laboratory

形式	命令	op	rs	rt	address/immediate
I	フィールド長	6	5	5	16
	beq	4	入力1	入力2	分岐先
	bne	5	入力1	入力2	分岐先
	addi	8	入力	出力	即値（符号付き）
	addiu	9	入力	出力	即値（符号なし）
	slti	10	入力	出力	即値（符号付き）
	sltiu	11	入力	出力	即値（符号なし）
	andi	12	入力	出力	即値（符号なし）
	ori	13	入力	出力	即値（符号なし）
	xori	14	入力	出力	即値（符号なし）
	lui	15	0	出力	即値（符号なし）
	lw	35	入力	出力	アドレス
	sw	43	入力	出力	アドレス

形式	命令	op	target address
J	フィールド長	6	26
	j	2	ジャンプ先
	jal	3	ジャンプ先