CSL
Computer Systems Laboratory

情報科学演習
プロセッサ演習
～第５週～

情報科学科
コンピュータシステム研究室

乗算命令

例1） mul $t0,$s1,$s2

011100(28)	10001(17)	10010(18)	01000(8)	00000(0)	000010(2)
6bit	5bit	5bit	5bit	5bit	6bit
32bit					
オペコード	**第1ソースオペランド**	**第2ソースオペランド**	**結果オペランド**	**シフト量**	**機能コード**
op	rs	rt	rd	shamt	funct

乗算（オーバーフロー無し）

$t0 ← $s1 * $s2

32bit × 32bit ＝ 64bit のうち、
下位32bitの結果のみを格納

※ hiレジスタ、loレジスタにそれぞれ上位32bit、下位32bitを格納することも同時に行う（本演習では使用しない）

乗算プログラム

```
addi   $s1, $zero, 13
addi   $s2, $zero, 11
mul    $t0, $s1, $s2
```

```
addi   $s1, $zero, 13
addi   $s2, $zero, 11
add    $a0, $zero, $s1
add    $a1, $zero, $s2
call   mul
add    $t0, $zero, $v0
```

jal

mul命令はMarsでは実行可能

verilog版MIPSでは
mul命令は未実装

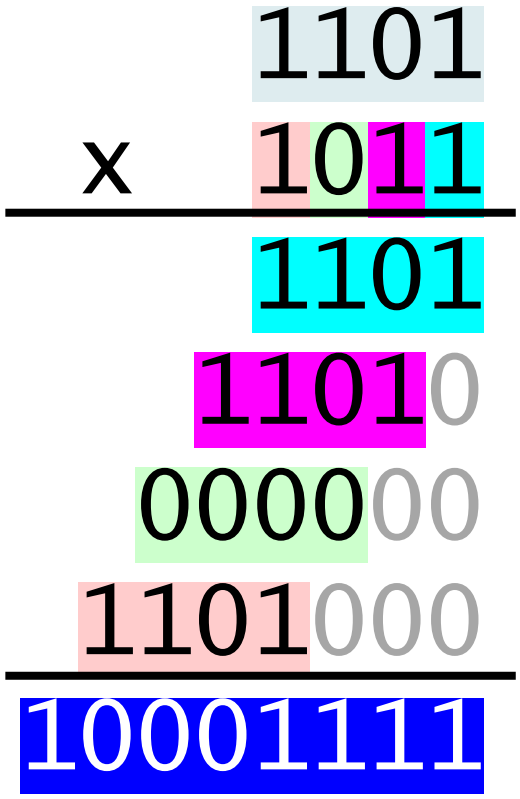
その１

定義済み命令を用いた
ビットシフト乗算で代用

その２

verilog版MIPSに
乗算命令を追加

ビットシフト乗算(原理)



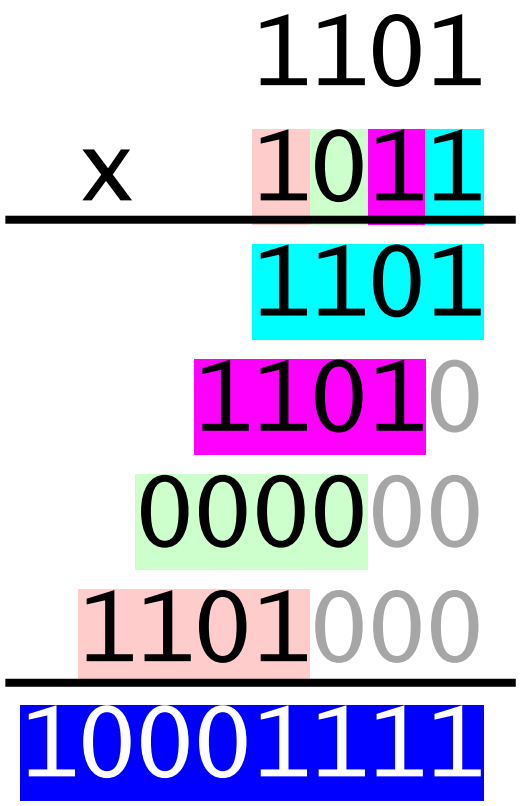

```
  00000000
+ 00001101   1011
  00001101

  00001101
+ 00011010   0101
  00100111

  00100111
+ 00000000   0010
  00100111

  00100111
+ 01101000   0001
  10001111
```

被乗数
左シフト

乗数
右シフト

積
乗数に応じて
被乗数を加算

ビットシフト乗算（C言語）




```
int mul( int a, int b )
{
  int y = 0;
  while( b != 0 ) {
    if( b & 1 == 1 ) {
      y += a;
    }
    a = a << 1;
    b = b >> 1;
  }
  return y;
}
```

被乗数
左シフト

乗数
右シフト

積
乗数に応じて
被乗数を加算

乗算命令の追加(verilog)

命令デコード　opから命令を特定

```
/* op: (00) R-format */
wire i_rfmt   = (op == 6'b000000);
/* op: (08) addi, (09) addiu */
/* fn: (20) add,  (21) addu */
wire i_add    = (op[5:1] == 5'b00100)
              | (fn[5:1] == 5'b10000) & i_rfmt;
/* fn: (22) sub, (23) subu */
wire i_sub    = (fn[5:1] == 5'b10001) & i_rfmt;
```

※ mips.v

乗算命令のデコード信号定義

命令	op	rs	rt	rd	shamt	funct
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
mul	28	入力1	入力2	出力	0	2

乗算命令の追加(verilog)

レジスタ指定　rs,rt,rdでレジスタ番号を指定

```
/* Source Register Numbers */
assign src_a = inst1[25:21];
assign src_b = (i_rfmt | i_brn | i_sw) ? inst1[20:16] : 5'h0;
/* Destination Register Number */
assign dst = (i_jal)  ? 5'b11111 : (i_rfmt) ? inst1[15:11]
           : (reg_wr) ? inst1[20:16] : 5’h0;
```

乗算命令のオペランドは3レジスタ

命令	op	rs	rt	rd	shamt	funct
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
mul	28	入力1	入力2	出力	0	2

乗算命令の追加(verilog)

レジスタ指定　rs,rt,rdでレジスタ番号を指定

```
/* Source Register Numbers */
assign src_a = inst1[25:21];
assign src_b = (i_rfmt | i_brn | i_sw) ? inst1[20:16] : 5'h0;
/* Destination Register Number */
assign dst = (i_jal)  ? 5'b11111 : (i_rfmt) ? inst1[15:11]
           : (reg_wr) ? inst1[20:16] : 5’h0;
```

乗算命令のオペランドは3レジスタ

命令	op	rs	rt	rd	shamt	funct
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
mul	28	入力1	入力2	出力	0	2

乗算命令の追加(verilog)

制御信号

alu_selは000と111が未使用（いずれかを乗算で使用）

```
/* Control Sygnals */
/* 001: addsub, 010: jal, 011: lui, 100: logic, 101: shift, 110: comp */
assign alu_sel[0] = i_lui   | i_shift | i_add | i_sub | i_lw | i_sw;
assign alu_sel[1] = i_lui   | i_comp  | i_jal;
assign alu_sel[2] = i_shift | i_comp  | i_logic;
```

ALUの第2入力はR形式と同じでレジスタ（条件追加）

```
assign src_sel_b = i_rfmt;
```

演算結果はレジスタに書き込み（条件追加）

```
assign reg_wr  = i_add | i_sub   | i_lw    | i_jal
               | i_lui | i_logic | i_shift | i_comp;
```

乗算命令の追加(verilog)

ALU

alu_selは000と111が未使用（いずれかに乗算の動作追加）

```
always@(posedge clk) begin
  case(alu_sel2)
    3'b001: alu_q3 <= daddsub;
    3'b010: alu_q3 <= dpc;
    3'b011: alu_q3 <= dlui;
    3'b100: alu_q3 <= dlogic;
    3'b101: alu_q3 <= dshift;
    3'b110: alu_q3 <= dslt;
    default: alu_q3 <= 32'h0;
  endcase
 end
```

第５週の課題（１）

- ビットシフト乗算のアセンブラプログラムを作成
  - MIPSコンパイラを使ってもOK
    - C言語のコンパイルは –O1 and/or –O2 で最適化
    - アセンブラプログラムでは引数を設定
    - 遅延分岐を使用
  - 直接アセンブラプログラムを書いてもOK
  - シミュレーションと実機で動作確認
- 乗算命令の追加
  - シミュレーションと実機で動作確認

再帰:階乗のプログラム

再帰

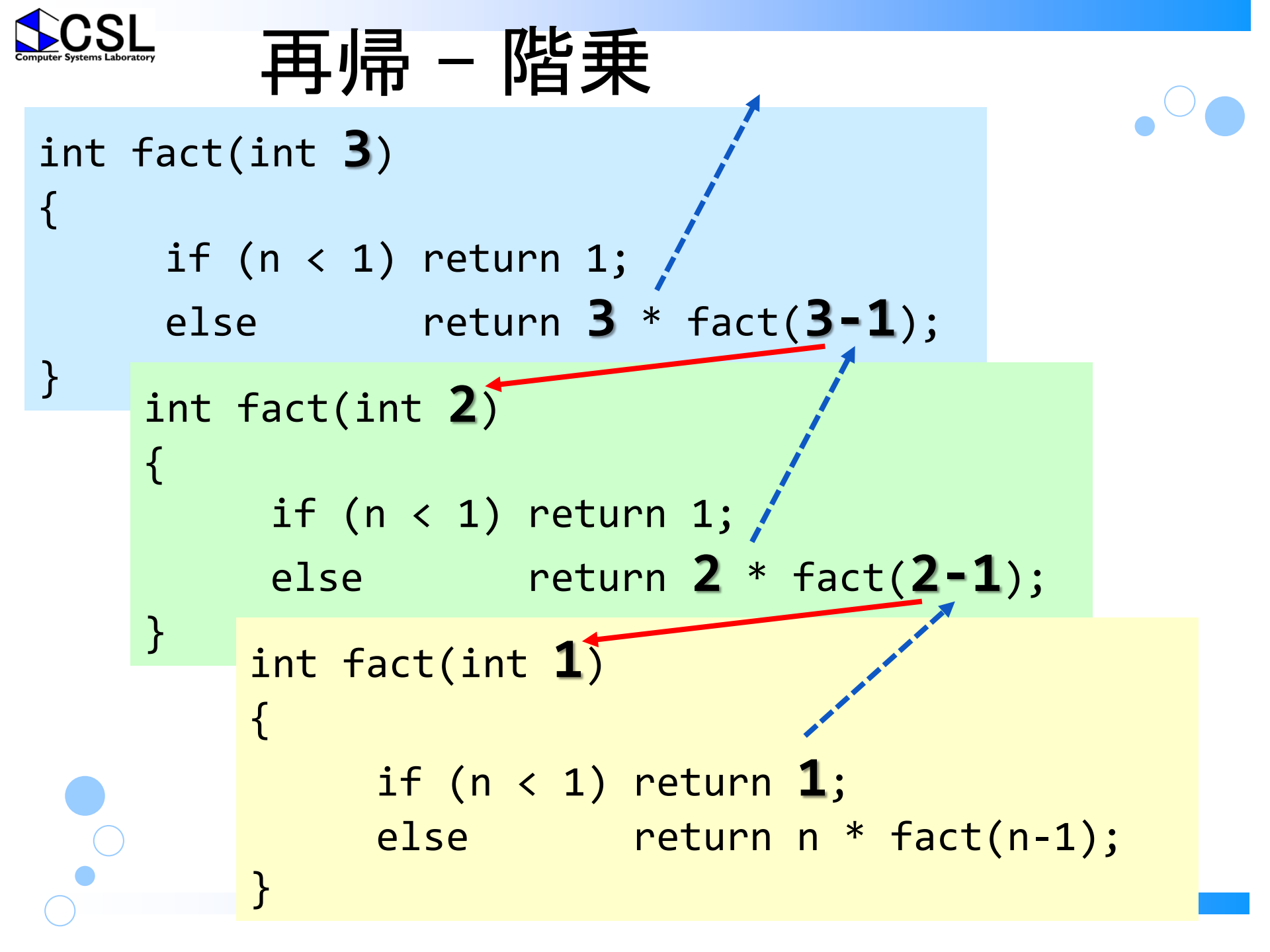
```
int fact(int n)
{
      if (n < 1) return  1 ;
      else       return  n * fact(n-1) ;
}
```

再帰 – 階乗

```
int fact(int 3)
{
      if (n < 1) return 1;
      else       return 3 * fact(3-1);
}
```

```
int fact(int 2)
{
      if (n < 1) return 1;
      else       return 2 * fact(2-1);
}
```

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



再帰(遅延分岐なし)

```
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
```

fact(0)

$ra(main)
$s0

fact(3)

$ra(main)
$s0(main)
$ra(fact)
$s0=3
$ra(fact)
$s0=2
$ra(fact)
$s0=1

再帰(遅延分岐あり)

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

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

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

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

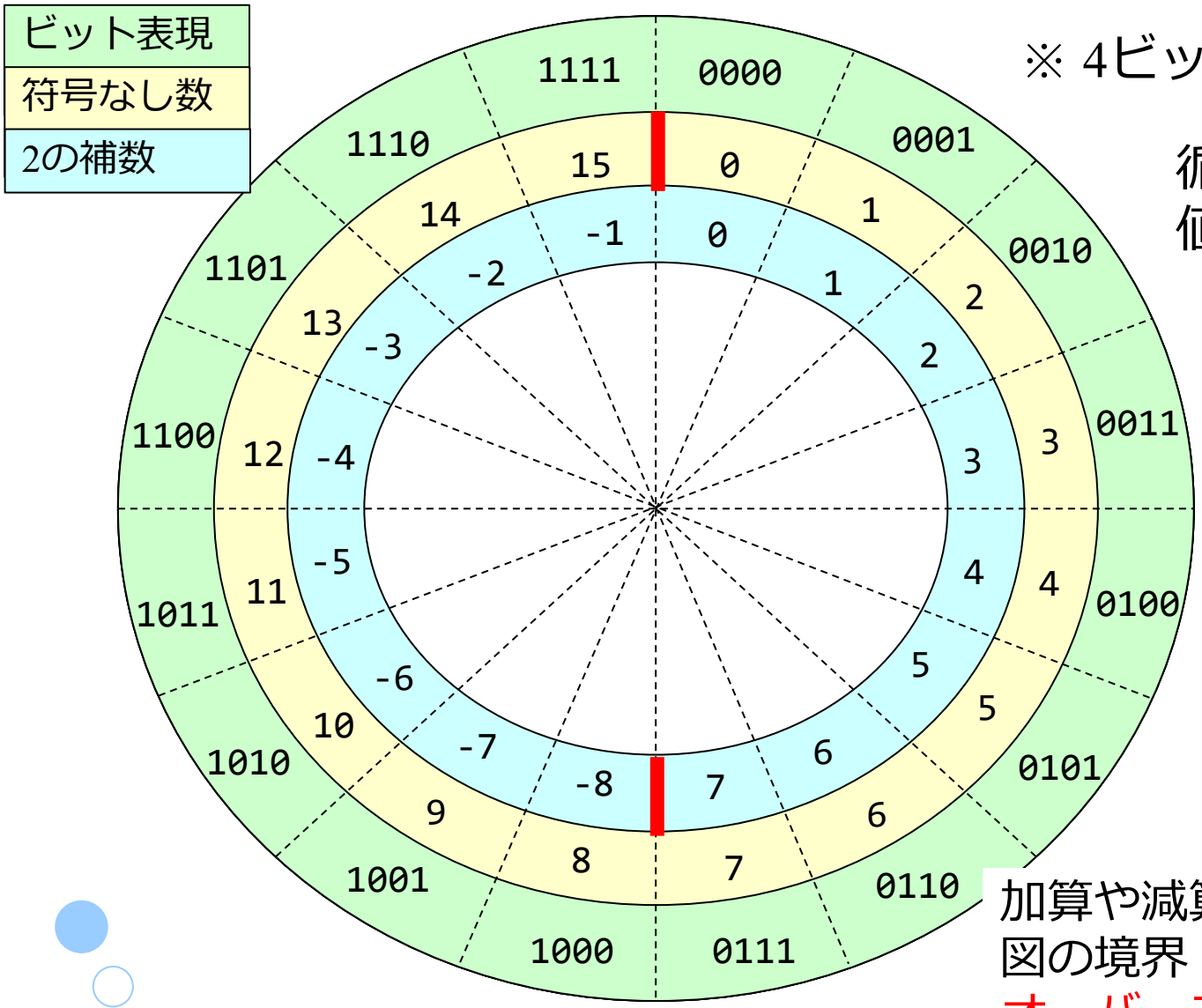
fact(0)

$ra(main)
$s0

fact(3)

$ra(main)
$s0(main)
$ra(fact)
$s0=3
$ra(fact)
$s0=2
$ra(fact)
$s0=1

符号なし数と2の補数



※ 4ビットの場合

循環するような
値のビット表現

加算や減算の演算結果が
図の境界（赤線）を超えると
オーバーフローとなる。
（正しい値を表現できない）

第５週の課題（２）

- 階乗のプログラムを実行
  - ビットシフト乗算と乗算命令のバージョンを比較
    - クロック数の違いは？
    - 結果の値が32ビットを超えた場合は？
- 注意事項
  - 乗算命令はオーバーフロー検出しない
  - 加算命令はMarsの場合、オーバーフロー検出する
    verilog版MIPSでは検出しない（Marsと動作が異なる）
  - スタックポインタ$spをverilog版MIPSで使うとき
    - 初期化：li $sp,0x10017ffc

再帰と末尾再帰

再帰

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

末尾再帰

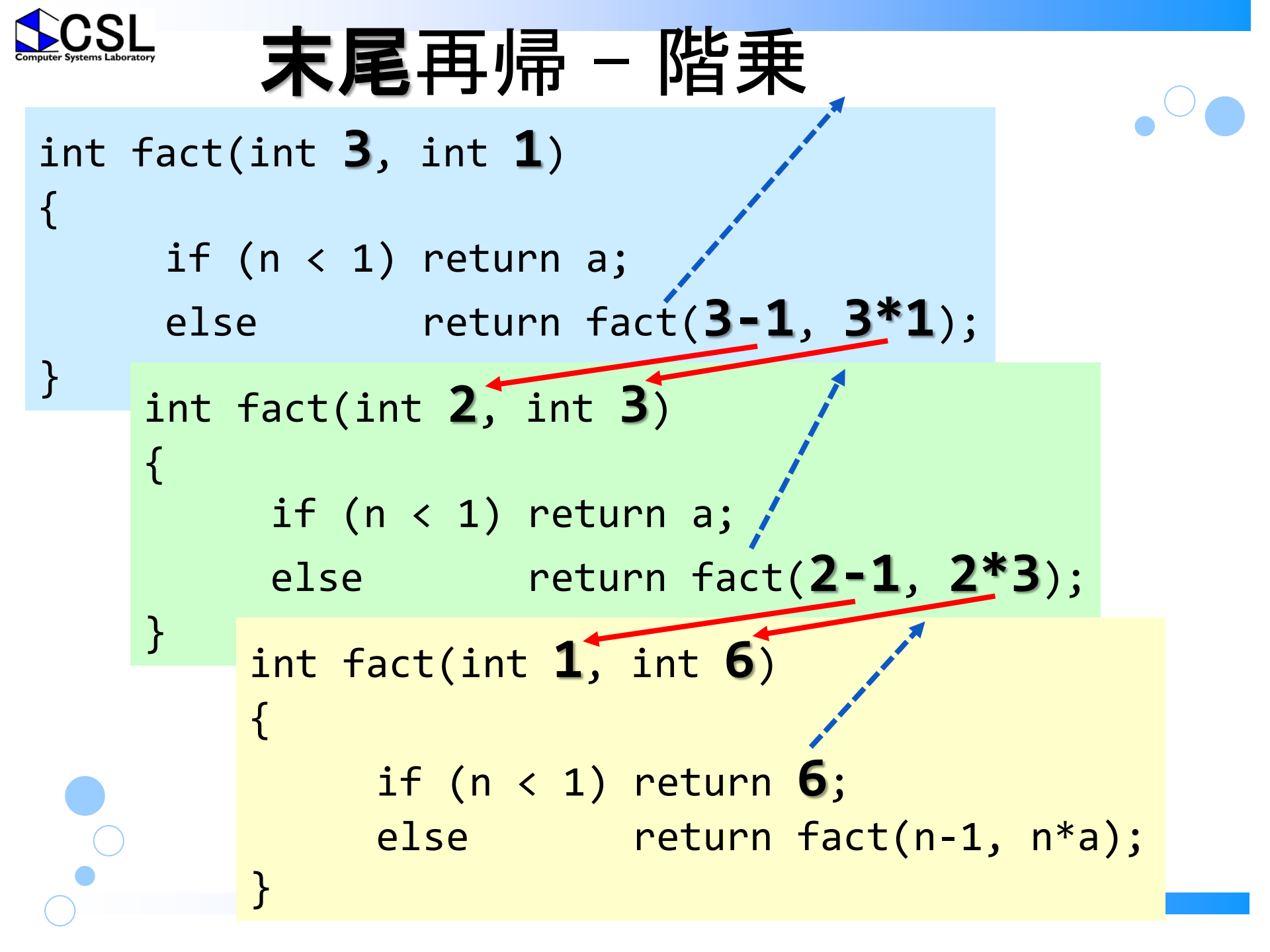
```
int fact(int n, int  a )
{
      if (n < 1) return  a ;
      else       return  fact(n-1,  n*a ) ;
}
```

末尾再帰 – 階乗

```
int fact(int 3, int 1)
{
      if (n < 1) return a;
      else       return fact(3-1, 3*1);
}
```

```
int fact(int 2, int 3)
{
      if (n < 1) return a;
      else       return fact(2-1, 2*3);
}
```

```
int fact(int 1, int 6)
{
      if (n < 1) return 6;
      else       return fact(n-1, n*a);
}
```



第５週の課題（３）

- 末尾再帰の階乗のアセンブラプログラムを作成
  - 乗算命令を用いたバージョンのみでOK
  - 直接書いてもMIPSコンパイラを使ってもOK
    - C言語のコンパイルは –O1 and/or –O2 で最適化
  - シミュレーションと実機で動作確認
  - 課題（２）の結果と比べて違いは？

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