末尾呼出し（tail call）
末尾再帰（tail recursion）

```
int sum(int n, int acc){
 if( n != 0 )
   return sum(n-1,acc+n);
 return acc;
}
```

```
sum:
 beq $a0,$zero,sum1
 add $a1,$a1,$a0
 addi $a0,$a0,-1
 beq $zero,$zero,sum
sum1:
 add $v0,$a1,$zero
 jr $ra
```


バイト転送命令と文字列コピー手続き

・バイト転送命令
- lb $t0,0($sp)
- sb $t0,0($sp)

```
void strcpy( char x[], char y[])
       // x[]のベースアドレス $a0
       // y[]のベースアドレス $a1
{
  int i; // $s0に割り当てる
  i = 0;
  while( (x[i]=y[i]) != 0 )
       i = i + 1;
}
```

"ABCD"

41	42	43	44	00

```
strcpy:
  addi $sp,$sp,-4
  sw $s0,4($sp)
  add $s0,$zero,$zero
L1:add $t1,$a1,$s0
  lb $t2,0($t1)
  add $t3,$a0,$s0
  sb $t2,0($t3)
  beq $t2,$zero,L2
  add $s0,$s0,1
  j L1
L2:lw $s0,4($sp)
  addi $sp,$sp,4
  jr $ra
```


効率の良いプログラム

```
strcpy:
  addi $sp,$sp,-4
  sw $s0,4($sp)
  add $s0,$zero,$zero
L1:add $t1,$a1,$s0
  lb $t2,0($t1)
  add $t3,$a0,$s0
  sb $t2,0($t3)
  beq $t2,$zero,L2
  add $s0,$s0,1
  j L1
L2:lw $s0,4($sp)
  addi $sp,$sp,4
  jr $ra
```

```
strcpy:
  addi $sp,$sp,-4
  sw $s0,4($sp)
  add $s0,$zero,$zero
  beq $zero,$zero,L2
L1:add $s0,$s0,1
L2:add $t1,$a1,$s0
  lb $t2,0($t1)
  add $t3,$a0,$s0
  sb $t2,0($t3)
  bne $t2,$zero,L1
  lw $s0,4($sp)
  addi $sp,$sp,4
  jr $ra
```


再帰関数：数字の文字列を数値に変換する

```
int num( char *p, int n )
{
    if ( *p == '¥0')  return n ;
    return num(p+1, n*10+*p-'0');
}
```

数字の文字列を数値に変換する

```
int num( char *p )
{
  int n ;
  n = 0 ;
  while( *p != '¥0' )
  {
    n = n * 10 + ( *p - '0' );
    p++;
  }
  return n ;
}
```

5

答え

```
num: add $t0,$zero,$a0
     add $t1,$zero,$zero
     beq $zero,$zero,num2
num1:sll $t2,$t1,1
     sll $t3,$t1,3
     add $t1,$t2,$t3
     lb $t3,0($t0)
     addi $t3,$t3,-48
     add $t1,$t1,$t3
     addi $t0,$t0,1
num2:lb $t2,0($t0)
     bne $t2,$zero,num1
     add $v0,$zero,$t1
     jr $ra
```

```
num: add $t0,$zero,$zero
     beq $zero,$zero,num2
num1:sll $t1,$t0,1
     sll $t2,$t0,3
     add $t0,$t1,$t2
     lb $t1,0($a0)
     addi $t1,$t1,-48
     add $t0,$t0,$t1
     addi $a0,$a0,1
num2:lb $t1,0($a0)
     bne $t1,$zero,num1
     add $v0,$zero,$t0
     jr $ra
```

6

オーバーフローの検出

・符号付き加算

```
addu $t0,$t1,$t2
xor $t3,$t1,$t2
slt $t3,$t3,$zero
bne $t3,$zero,NoOV
xor $t3,$t0,$t1
slt $t3,$t3,$zero
bne $t3,$zero,OV

NoOV: //オーバーフローなし

OV:   //オーバーフロー処理
```

・符号無し加算

```
addu $t0,$t1,$2
nor $t3,$t1,$zero
sltu $t3,$t3,$t2
bne $t3,$zero,OV
NoOV://オーバーフローなし


OV://オーバーフロー処理
```

7

2倍長（64ビット）加算　**dadd.s**

```
      .text
      .globl __start
__start:
      add $a0,$zero,0x40000000
      add $a1,$zero,0x00001234
      add $a2,$zero,0x3FFFFFFF
      add $a3,$zero,0x0
      jal dadd
      break 2
```

関数dadd：オーバーフロー検出を使って，64ビット同士の加算をする．

```
dadd:  add $v1,$zero,$zero
       addu $v0,$a0,$a2
       nor $t1,$a0,$zero
       sltu $t1,$t1,$a2
       beq $t1,$zero,No_OV
       add $v1,$v1,1
No_OV: add $v1,$v1,$a1
       add $v1,$v1,$a3
       jr $ra
```

8