fact関数の補足

■ 教科書の解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,L1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
L1: addi $a0,$a0,-1
    jal fact
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

■ 講義で示した解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
fact1:
    addi $a0,$a0,-1
    jal fact
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

■ 講義で示した解答の最適化

```
fact:
    slti $t0,$a0,1
    beq $t0,$0,fact1
    ori $v0,$0,1
    jr $ra
fact1:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    addi $a0,$a0,-1
    jal fact
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

j命令とjal命令　擬似直接アドレッシング

■ 1命令は4バイト固定である
→ジャンプ先アドレスは4番地おきである
→ジャンプ先アドレスの下位2ビットは必ず0
→アドレスフィールドには4分の1の値を格納すればよい（例 10000ではなく2500）

■ 10000番地に無条件ジャンプ

```
j 10000
```

2	2500
6bits	26bits

■ 10000番地にジャンプ&リンク

```
jal 10000
```

3	2500
6bits	26bits

- $ra=PC+4; goto 10000

■ 擬似直接アドレッシング

- MIPSのアドレスは32ビット
- 命令では26+2ビットしか指定していない
- 4ビット不足している
  - 不足しているので「擬似」

■ PC=JumpAddr

```
JumpAddr={PC(31:28),address,2b'0}
```

4ビット　26ビット　2ビット

bne/beq命令　PC相対アドレッシング

■ 1命令は4バイト固定である
→相対番地でも4バイト単位で分岐する
→アドレスフィールドには4分の1の値を格納すればよい

■ 条件分岐(xx : ne or eq)

```
bxx $s0,$s1,Exit
```

5	16	17	Exit
6bits	5bits	5bits	16bits

■ 条件が成立したとき　PC=PC+4+(−2^{15}~2^{15}−1)×4
条件が成立しないとき PC=PC+4

■ $s1と$s2が等しいとき，PC相対で100番地先に飛ぶ

- beq $s1,$s2,25 #if($s1==$s2)goto PC+4+100

■ "PC+4"は何か？

- PC=**PC+4**+(−2^{15}~2^{15}−1)×4
- PC=**PC+4**

■ "PC+4"は次の命令のアドレスである

whileループ

■ Cコード

```
while(save[i]==k)
    i += 1;
```

- i,kは$s3,$s5
- saveのベースアドレスは$s6

■ MIPSコード

```
Loop:sll $t1,$s3,2
     add $t1,$t1,$s6
     lw $t0,0($t1)
     bne $t0,$s5,Exit
     addi $s3,$s3,1
     j Loop
Exit:
```

080000	0	0	19	9	2	0	sll $t1,$s3,2
080004	0	9	22	9	0	32	add $t1,$t1,$s6
080008	35	9	8	0			lw $t0,0($t1)
080012	5	8	21	2			bne $t0,$s5,Exit
080016	8	19	19	1			addi $s3,$s3,1
080020	2	2000					j Loop
080024							

whileループの補足（1）

■ MIPSコード

```
Loop:sll $t1,$s3,2
     add $t1,$t1,$s6
     lw $t0,0($t1)
     bne $t0,$s5,Exit
     addi $s3,$s3,1
     j Loop
Exit:
```

■ MIPSコード

```
Loop:sll $t1,$s3,2
     add $t1,$t1,$s6
     lw $t0,0($t1)
     bne $t0,$s5,Exit
     addi $s3,$s3,1
     beq $zero,$zero,Loop
Exit:
```

080000	0	0	19	9	2	0	sll $t1,$s3,2
080004	0	9	22	9	0	32	add $t1,$t1,$s6
080008	35	9	8	0			lw $t0,0($t1)
080012	5	8	21	2			bne $t0,$s5,Exit
080016	8	19	19	1			addi $s3,$s3,1
080020	4	0	0	-6			beq $zero,$zero,Loop
080024							

whileループの補足（2）

■ 条件が成立したとき　PC=PC+4+(−2^{15}~2^{15}−1)×4
　条件が成立しないとき PC=PC+4

■ 080012　bne $t0,$s5,**2**
　▸ 分岐先PC = 080012+4+**2**×4 = 080024

■ 080020　beq $zero,$zero,**-6**
　▸ 分岐先PC = 080020+4+**(-6)**×4 = 080000

■ "PC+4"は次の命令のアドレスである
　▸ PC=**PC+4**+(−2^{15}~2^{15}−1)×4
　▸ PC=**PC+4**

■ (−2^{15}~2^{15}−1)×4は何か？
　▸ 正：次の命令から何命令後か？
　▸ 負：次の命令から何命令前か？
　▸ -1：次の命令から1命令前
　　→ 同じ命令を実行し続ける

bne	beq									
-4	-6	080000	0	0	19	9	2	0	sll $t1,$s3,2
-3	-5	080004	0	9	22	9	0	32	add $t1,$t1,$s6
-2	-4	080008	35	9	8	0			lw $t0,0($t1)
-1	-3	080012	5	8	21	2			**bne $t0,$s5,Exit**
0:PC+4	-2	080016	8	19	19	1			addi $s3,$s3,1
1	-1	080020	4	0	0	-6			**beq $zero,$zero,Loop**
2	0:PC+4	080024							

より遠くへの分岐

■ L1の値が16ビットに収まらないとき

```
beq $s0,$s1,L1
```

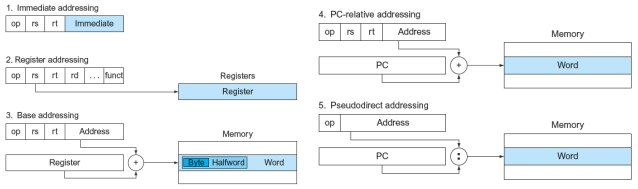
■ j命令と組み合わせれば良い．

```
    bne $s0,$s1,L2
    j L1
L2:
```

MIPSアドレシングモード

1. 即値アドレッシング
2. レジスタ・アドレッシング
3. ベース・アドレッシング
4. PC相対アドレッシング
5. 擬似直接アドレッシング
   ‣ 32ビットの指定ならば「直接」，26ビットしか指定しているから「擬似直接」

PC=JumpAddr
JumpAddr={PC{31:28},**address**,2b'0}



マシン・コードの復号

■ 次の機械語の命令に相当するアセンブリ言語の命令は何か？

$(00af8020)_{16}$

■ 16進数を2進数に変換する

0000 0000 1010 1111 1000 0000 0010 0000

■ OPコードから命令の種類を判定する

(000000)はR形式の命令である

■ 命令はR形式なので，機械語をR形式に直す．

000000	00101	01111	10000	00000	100000
op	rs	rt	rd	shamt	funct

```
add $s0,$a1,$t7
```

オーバーフローの検出（176頁）

■ 符号付き加算でのオーバーフロー

```
    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
```

■ 符号無し加算でのオーバーフロー

```
    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
```

オーバフロー例外を発生させないため
add　ではなくて　addu
を使う．

オーバーフロー　４ビット符号付き整数同士の加算

■異符号同士の加算では発生しない

■同符号同士の加算で発生する場合がある．

- 正＋正＝負
- 負＋負＝正

		-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
		1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
-8	1000	OV	OV	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111
-7	1001	OV	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000
-6	1010	OV	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001
-5	1011	OV	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010
-4	1100	OV	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011
-3	1101	OV	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100
-2	1110	OV	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101
-1	1111	OV	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110
0	0000	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
1	0001	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV
2	0010	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV
3	0011	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV
4	0100	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV
5	0101	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV
6	0110	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV	OV
7	0111	1111	0000	0001	0010	0011	0100	0101	0110	0111	OV	OV	OV	OV	OV	OV	OV

符号付き加算におけるオーバーフローの検出

■符号付き加算でのオーバーフロー

```
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
```

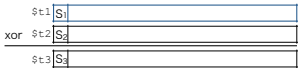
```
NoOV: //オーバーフローなし

OV:   //オーバーフロー処理
```

■$t1と$t2が異符号ならばオーバーフローしない

■xorで同符号か異符号かを調べることができる

- $t1と$t2が同符号ならば
  S_3=0となり，$t3は0以上の正数である
- $t1と$t2が異符号ならば
  S_3=1となり，$t3は負数である



■$t1と$t2が同符号ならば，加算結果$t0と$t1（$t2）の符号が異なればオーバーフローが発生している．

オーバーフロー　４ビット符号無し整数同士の加算

■加算の結果が$(1111)_2$を超えるとき

■1011+0110=10001

■検出条件

- X+Y>10000

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0001	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV
0010	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV
0011	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV
0100	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV
0101	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV
0110	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV
0111	0111	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV
1000	1000	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV
1001	1001	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV
1010	1010	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1011	1011	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1100	1100	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1101	1101	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1110	1110	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV
1111	1111	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV	OV

符号無し加算におけるオーバーフローの検出

■符号無し加算でのオーバーフロー

```
    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
```

■ビット反転（NOT）はnor命令で計算できる

```
nor $t3,$t1,$zero # $t3= NOT $t1
```

■ビット反転した値は，次のように表すことができる

```
$t3= NOT $t1 = 2^32-1 - $t1
```

■sltu $t3,$t3,$t2の意味は次の通り．

```
$t3          < $t2

2^32-1 - $t1  < $t2

2^32-1        < $t2+$t1
```

32ビットの最大値

■次の条件が成立すれば，オーバーフローが発生している

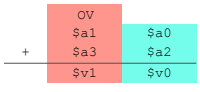
```
2^32-1 < $t2+$t1
```

２倍長の加算 dadd.s

■64ビット同士の加算をする

■上位32ビットと下位32ビットに分けて考える

- 下位32ビットは符号無し加算で，オーバーフローを検出する
- 上位32ビットは符号付き加算で，下位32ビットの加算からオーバーフローがあれば加算結果に１を加える．




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

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
```