コンピュータアーキテクチャ論　演習３回

1. 行列積の計算

ソースコード

```
    .data
A:  .word 0
    .word 1
    .word 0
    .word 0
    .word 2
    .word 0
    .word 0
    .word 0
    .word 0
    .word 0
    .word 0 .word 3
    .word 0
    .word 0
    .word 4
    .word 0
B:  .word 1
    .word 2
    .word 3
    .word 4
    .word 5
    .word 6
    .word 7
    .word 8
    .word 9
    .word 10
    .word 11
    .word 12
    .word 13
    .word 14
    .word 15
    .word 16
C:  .space 64
N:  .word 4
M:  .word 4

    .text
main:
    la $8, A
```

```
    la $9, B
    la $10, C
    lw $11, N
    lw $12, M
    add $13, $0, $0 # sum
    add $14, $0, $0 # init i
    add $15, $0, $0 # init j
    add $24, $0, $0 # init k

out_loop:
    beq $14, $11, loopend # i == N
    add $15, $0, $0 # init j
    in_loop:
        beq $15, $12, out_inc # j == M
        add $13, $0, $0 # sum = 0
        add $24, $0, $0 # init k
        loop:
            beq $24, $12, in_loopend # k == M

            # m1のアドレスを計算
            add $4, $14, $0
            addi $5, $0, 4
            jal MUL # i * 4
            add $4, $2, $24 # i * 4 + k
            addi $5, $0, 4
            jal MUL # (i * 4 + k) * 4
            add $8, $8, $2 # addr + (i * 4 + k) * 4

            # m2のアドレスを計算
            add $4, $24, $0
            addi $5, $0, 4
            jal MUL # k * 4
            add $4, $2, $15 # k * 4 + j
            addi $5, $0, 4
            jal MUL # (k * 4 + j) * 4
            add $9, $9, $2 # addr + (k * 4 + j) * 4

            lw $4, 0($8) # m1
            lw $5, 0($9) # m2
            jal MUL # m1 * m2
            add $13, $13, $2 # sum += m1 * m2

            addi $24, $24, 1
            la $8, A
            la $9, B
            j loop

in_loopend:
    # 保存先のアドレスを計算
```

```
    add $4, $14, 0
    addi $5, $0, 4
    jal MUL
    add $4, $2, $15
    addi $5, $0, 4
    jal MUL
    add $10, $10, $2

    sw $13, 0($10) # 結果を保存
    addi $15, $15, 1
    la $10, C
    j in_loop

out_inc:
    addi $14, $14, 1
    j out_loop

loopend:

exit: j exit

MUL:
    addi $16, $0, 1 # mask
    addi $17, $0, 0 # i
    addi $18, $0, 16 # N
    addi $2, $0, 0 # ans

MUL_loop:
    beq $17, $18, MUL_exit
    and $19, $4, $16
    beq $19, $0, MUL_inc
    addu $2, $2, $5
    j MUL_inc

MUL_inc:
    addi $17, $17, 1
    addu $16, $16, $16
    addu $5, $5, $5
    j MUL_loop

MUL_exit:
jr $ra # サブルーチンの呼び出し元に戻る
```

結果

Data Segments				
[0x00005060]	0x00000009	0x0000000a	0x0000000b	0x0000000c
[0x00005070]	0x0000000d	0x0000000e	0x0000000f	0x00000010
[0x00005080]	0x00000005	0x00000006	0x00000007	0x00000008
[0x00005090]	0x00000002	0x00000004	0x00000006	0x00000008
[0x000050a0]	0x00000027	0x0000002a	0x0000002d	0x00000030
[0x000050b0]	0x00000024	0x00000028	0x0000002c	0x00000030
[0x000050c0]	0x00000004	0x00000004	0x00000000	0x00000000
[0x000050d0]...[0x00025000]	0x00000000			

2. 再帰による階乗

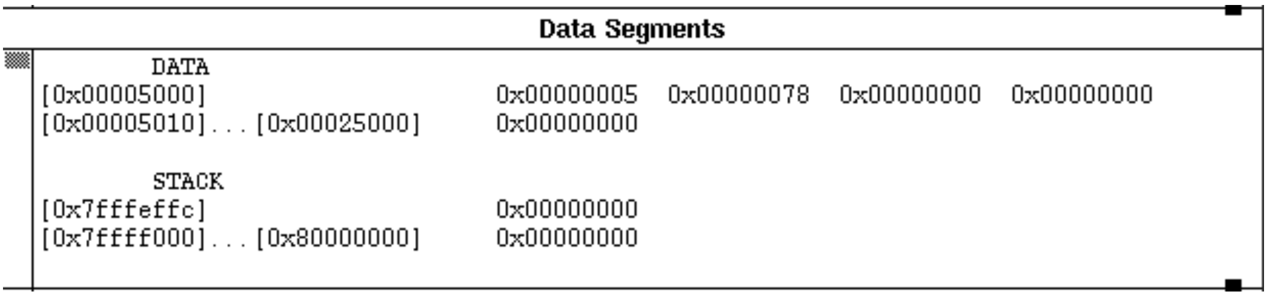
ソースコード

```
    .data
N:  .word 5
FN: .word 0

.text
main:
    lw $a0, N
    jal fact
    sw $v0, FN
    exit: j exit

fact:
    # スタックにraとa0を保存
    addi $sp, $sp, -8 # スタックポインタを-8する（２つ分入れるため）
    sw $ra, 4($sp)
    sw $a0, 0($sp)

    # a0が1より小さいか確認
    slti $t0, $a0, 1
    beq $t0, $0, L1 # 1より大きかったらL1に行く

    addi $v0, $0, 1
    addi $sp, $sp, 8
    jr $ra

L1:
    addi $a0, $a0, -1
    jal fact
    lw $a0, 0($sp)
    add $a1, $0, $v0
    jal MUL # 掛け算
    lw $ra, 4($sp) # 掛け算で$raの値が変わるのでここでスタックからロードする
    addi $sp, $sp, 8 # スタックのアドレスを戻す
    jr $ra

MUL:
```

```
    addi $s0, $0, 1 # mask
    addi $s1, $0, 0 # i
    addi $s2, $0, 16 # N
    addi $s4, $0, 0 # ans

MUL_loop:
    beq $s1, $s2, MUL_exit
    and $s3, $a0, $s0
    beq $s3, $0, MUL_inc
    addu $s4, $s4, $a1
    j MUL_inc

MUL_inc:
    addi $s1, $s1, 1
    addu $s0, $s0, $s0
    addu $a1, $a1, $a1
    j MUL_loop

MUL_exit:
    add $v0, $0, $s4
jr $ra
```

結果

N = 5



N = 8