

```

Proc2:  addi  $29, $29, -4
        sw   $16, 0($29)

        muli $8, $5, 4
        add  $8, $4, $8

        lw   $16, 0($8)
        lw   $9, 4($8)

        sw   $9, 0($8)
        sw   $16, 4($8)

        lw   $16, 0($29)
        addi $29, $29, 4

        jr   $31

```

```

Proc1:  addi  $29, $29, -20
        sw   $16, 0($29)
        sw   $17, 4($29)
        sw   $18, 8($29)
        sw   $19, 12($29)
        sw   $31, 16($29)

        move $16, $4
        move $17, $5

        add  $18, $0, $0
for1tst: slt   $8, $18, $17
        beq  $8, $0, exit1

        addi $19, $18, -1
for2tst: slti  $8, $19, 0
        bne  $8, $0, exit2
        muli $9, $19, 4
        add  $10, $16, $9
        lw   $11, 0($10)
        lw   $12, 4($10)
        slt  $8, $12, $11
        beq  $8, $0, exit2

        move $4, $16
        move $5, $19
        jal  Proc2

        addi $19, $19, -1
        j   for2tst

exit2:  addi $18, $18, 1
        j   for1tst

exit1:  lw   $16, 0($29)
        lw   $17, 4($29)
        lw   $18, 8($29)
        lw   $19, 12($29)
        lw   $31, 16($29)
        addi $29, $29, 20

        jr   $31

```

## Notes

Register 0 always has the value of 0.

A subroutine call is made using a jump-and-link instruction (jal). The return address is placed in register 31, not on the stack because there is no implicit stack pointer defined in the instruction set architecture.

For subroutines, the following conventions are used:

Register Name	Register Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0,v1	2,3	Function return results
a0-a3	4-7	Arguments 1-4
t0-t7	8-15	Temporary (not preserved across procedure call)
s0-s7	16-23	Saved temporary (preserved across procedure call)
t8,t9	24,25	Temporary (not preserved across procedure call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by procedure call)

Remember that these are conventions set by the compiler writer. Except for register 31 being used to hold a return address when a jal instruction is executed, there are no side-effects or special meanings assigned to the registers in the instruction set definition.

The procedure call stack grows from larger addresses to small addresses.

```
int v[10000];
```

```
sort (int v[], int n);
{
    int i, j;
    for (i = 0; i < n; i = i+1) {
        for (j = i-1; j >= 0 && v[j] > v[j+1]; j = j-1) {
            swap(v,j);
        }
    }
}
```

```
swap (int v[], int k);
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```