factorial MIPS code, with registers

```
factorial:
        sw    $fp, -4($sp)
        addiu $fp, $sp, -4
        sw    $ra, -4($fp)
        addiu $sp, $fp, -8
        ori   $t0, $0, 1
        sw    $t0, -8($fp)
        lw    $t0, 4($fp)
        ori   $t1, $0, 0
        slt   $t0, $t1, $t0
        beq   $t0, $0, l1
        j     l0
l0:     lw    $t0, 4($fp)
        lw    $t1, 4($fp)
        ori   $t2, $0, 1
        subu  $t1, $t1, $t2
        addiu $sp, $sp, -4
        sw    $t0, 0($sp)
        addiu $sp, $sp, -4
        sw    $t1, 0($sp)
        jal   factorial
        or    $t1, $0, $v0
        lw    $t0, 0($sp)
        addiu $sp, $sp, 4
        mult  $t0, $t1
        mflo  $t0
        sw    $t0, -8($fp)
l1:     lw    $t0, -8($fp)
        or    $v0, $0, $t0
        lw    $ra, -4($fp)
        addiu $sp, $fp, 8
        lw    $fp, 0($fp)
        jr    $ra
```

Optimised `factorial` MIPS code

```
factorial:
        sw    $fp, -4($sp)
        addiu $fp, $sp, -4
        sw    $ra, -4($fp)
        addiu $sp, $fp, -8
        ori   $t0, $0, 1
        sw    $t0, -8($fp)
        lw    $t0, 4($fp)
        blez  $t0, l1
l0:     lw    $t0, 4($fp)
        addiu $t1, $t0, -1
        sw    $t0, -4($sp)
        addiu $sp, $sp, -8
        sw    $t1, 0($sp)
        jal   factorial
        lw    $t0, 0($sp)
        addiu $sp, $sp, 4
        mult  $t0, $v0
        mflo  $t0
        sw    $t0, -8($fp)
l1:     lw    $v0, -8($fp)
        lw    $ra, -4($fp)
        addiu $sp, $fp, 8
        lw    $fp, 0($fp)
        jr    $ra
```

Optimised `factorial` MIPS code (2)

```
factorial:
        sw    $fp, -4($sp)
        addiu $fp, $sp, -4
        sw    $ra, -4($fp)
        addiu $sp, $fp, -8
        ori   $t0, $0, 1
        sw    $t0, -8($fp)
        lw    $t0, 4($fp)
        blez  $t0, l1
        addiu $t0, $t0, -1
        addiu $sp, $sp, -4
        sw    $t0, 0($sp)
        jal   factorial
        lw    $t0, 4($fp)
        mult  $t0, $v0
        mflo  $t0
        sw    $t0, -8($fp)
l1:     lw    $v0, -8($fp)
        lw    $ra, -4($fp)
        addiu $sp, $fp, 8
        lw    $fp, 0($fp)
        jr    $ra
```

Optimised `factorial` MIPS code (3)

```
factorial:
        sw    $fp, -4($sp)
        addiu $fp, $sp, -4
        sw    $ra, -4($fp)
        addiu $sp, $fp, -12
        sw    $s0, -12($fp)
        ori   $s0, $0, 1
        lw    $t0, 4($fp)
        blez  $t0, l1
        addiu $t0, $t0, -1
        addiu $sp, $sp, -4
        sw    $t0, 0($sp)
        jal   factorial
        lw    $t0, 4($fp)
        mult  $t0, $v0
        mflo  $s0
l1:     or    $v0, $0, $s0
        lw    $s0, -12($fp)
        lw    $ra, -4($fp)
        addiu $sp, $fp, 12
        lw    $fp, 0($fp)
        jr    $ra
```

Optimisations (1)

Constant propagation

Try to replace uses of a register/variable known to contain a constant value by the constant

Copy propagation

Try to replace uses of a register/variable known to be a copy of another register/variable by the latter

Peephole optimisation

Using a sliding window, identify code patterns and replace them by better alternatives

Dead-code elimination

Remove code never executed or that produces a value which is never used and that has no side effects

Optimisations (2)

Constant folding

Replace expressions involving constants by their value

Strength reduction

Replace an operation by a less costly semantically equivalent operation

Loop unrolling

Try to optimise code resulting from unrolling a loop

Lazy code motion

Move code so that it is no longer redundantly executed

Used to remove loop-invariant code from within a loop

Code hoisting

Reduce code size by folding code common to several execution paths

Inline substitution

Replace a function call by the code of the function called

Use-def and def-use chains

Reaching definitions

Definition d of x reaches operation i if there is a path from d to i, in the CFG, that does not define x

d is a reaching definition of x at i

use-def chain

For a use of x, the list of reaching definitions of x

def-use chain

For a definition of x, the list of all its possible uses

Use-def chains and def-use chains support many of the preceding optimisations