Oversætter - Week 4

Exercise 1

1a

Translate expression $gcd(x+y, y+1)*2$ to intermediate code. Place the result in $t0$ and assume that $vtable = [x \to v, y \to w], ftable = [gcd \to \_GCD\_]$.

The intermediate code looks like this

```
t1 := 1
t2 := v
t3 := w
t4 := t2+t3
t5 := t4+t1
t6 := CALL _GCD_(t4, t5)
t7 := 2
t0 := t6*t2
```

1b

Translate the following statement, i.e., Stat in grammar, to intermediate code, and then to MIPS code:

```
 while (b != 0) and (a/b != 0) {
     if b < a then { a := a - b }
              else { b := b - a }
 }
```

where initially $vtable = [a- > v, b- > w]$

In intermediate code:

```
t1:= v
t2:= w
LABEL while
    t3 := t2 < 0
    t4 := 0 < t2
    t5 := t3 || t4
    IF t5 THEN bNotZero ELSE whileFalse
    LABEL bNotZero
        t3 := t1 / t2
        t4 := t3 < 0
        t6 := 0 < t3
        t7 := t4 || t6
        IF t5 && t7 THEN whileTrue ELSE whileFalse
        LABEL whileTrue
            t0 := t2 < t1
            IF t0 THEN ifTrue ELSE ifFalse
            LABEL ifTrue
                t1 := t1 - t2
                GOTO while
            LABEL ifFalse
                t2 := t2 - t1
                GOTO while
LABEL whileFalse
```

And in MIPS code and assuming $a = v_0$ and $b = v_1$:

```
add     $t1, $v0, $zero
add     $t2, $v1, $zero

while:
beqz    $t2, end
divu    $t0, $t1, $t2
beqz    $t0, end
slt     $t0, $t2, $t1
beqz    $t0, ifTrue
sub     $t2, $t2, $t1
j       while

ifTrue:
sub     $t1, $t1, $t2
j       while

end:
```

1c

Make pattern/replacement pairs for each of the following intermediate-language instructions:

$$(i) rd := rs = rt$$
$$(ii) rd := !r$$

For (i) we get (with $rd = a_0$, $rs = a_1$ and $rt = a_2$)

```
slt $t0, $a0, $a1
slt  $t1, $a1, $a0
or $t2, $t0, $t1
xori $a0, $t2, 1
```

for (ii) we get ($rd = a0$, $r = a1$)

```
slt $t0, $a1, $zero
slt  $t1, $zero, $a1
or $t2, $t0, $t1
xori $a0, $t2, 1
```

Exercise 2

2a

Translate the call map(f, x) in Paladim (using while loops)

```
function map(f : function, x : array of array of int)
var i : int;
    j : int;
begin
    i := 0;
    j := 0;
    while (i < len(0,x)) do
      begin
        while (j < len(1,x)) do
          begin
            x[i][j] := f(x[i][j]);
            j := j +1;
          end;
        i := i + 1;
      end;
end;
```

2b

Implement the call map(f, x) in Mips. Assume

vtable is [x → regx]
ftable is [f → "_f_"] and HP is the heap pointer.

You may also assume that you have a Mips instruction that calls a function, e.g., Mips.CALL(f, [re]).

```
addi    $t1, HP, 16
lw      $t2, 0(HP)
lw      $t3, 4(HP)
mul     $t4, $t2, $t3
sll     $t4, $t4, 2
beq     $t4, $zero, end
add     $t4, $t4, HP
```

```
while:
lw      $t5, 0($t0)
Mips.CALL(_f_, $t5)
addi    $t5, $t5, 4
slt     $t0, $t5, $t4
bne     $t0, $zero, while

end:
```

2c

Which version would be more efficient and Why?

There is more branching in the paladim code because of the two while-loops, whereas the in the MIPS code there is only 1 while loop. Therefore the MIPS code is faster.