Foundations of Programming Languages

Implementing Iterative Control Structures

Prof. Dr. Christoph Reichenbach

Fachbereich 12 / Institut für Informatik

29. Oktober 2014

Implementing Loops

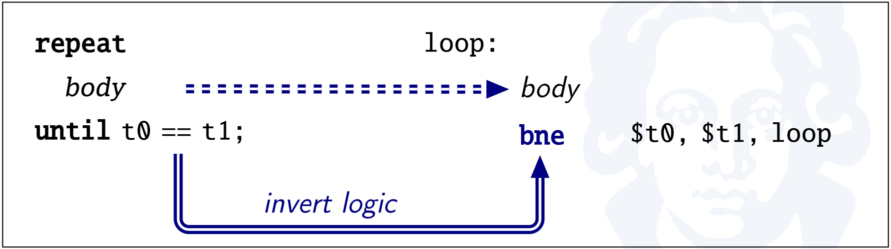
- Compilers translate loops into:
  - tests
  - branches
- Some complications
- Some opportunities for optimisation

Logical Loops with Branches

```
repeat
  body
until t0 == t1;
```

Other implementation options exist

Logical Loops with Branches



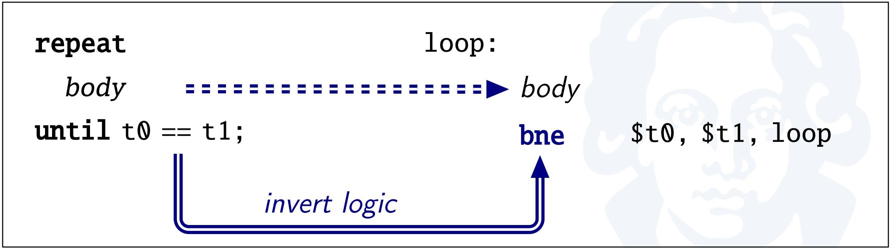
```
repeat
  body
until t0 == t1;
```

```
loop:
body
bne    $t0, $t1, loop
```

Other implementation options exist

Logical Loops with Branches

```
                          loop:
while t0 == t1 do
  body
done
```

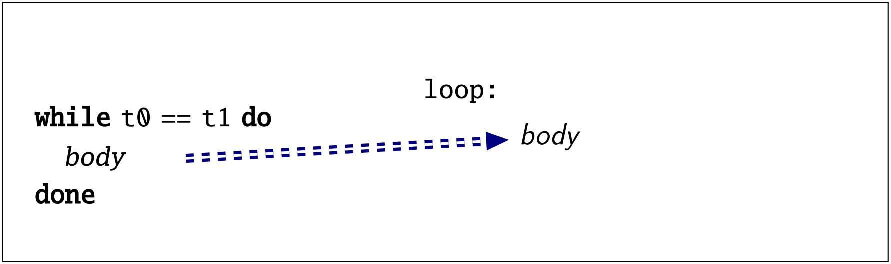


```
repeat                    loop:
  body    ================> body
until t0 == t1;             bne      $t0, $t1, loop
```

invert logic

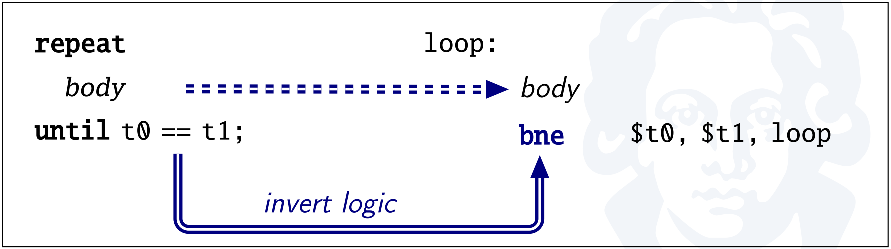
Other implementation options exist

Logical Loops with Branches



```
while t0 == t1 do
  body
done
```

```
loop:
  body
```



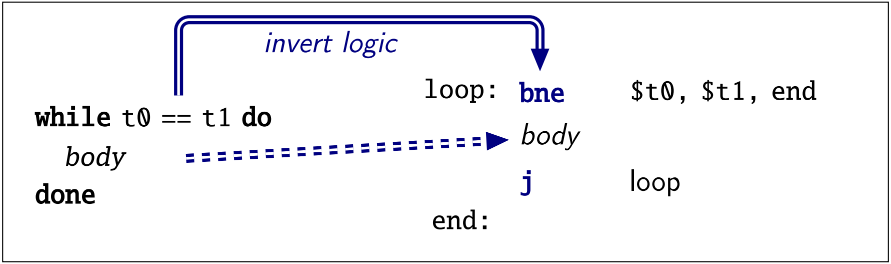
```
repeat
  body
until t0 == t1;
```

```
loop:
  body
  bne     $t0, $t1, loop
```

invert logic

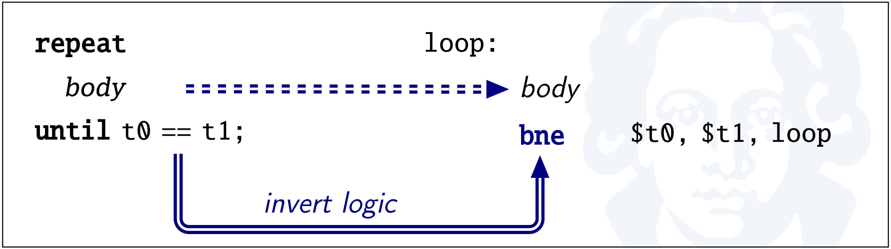
Other implementation options exist

Logical Loops with Branches



```
while t0 == t1 do
   body
done
```

```
loop: bne    $t0, $t1, end
      body
      j      loop
end:
```

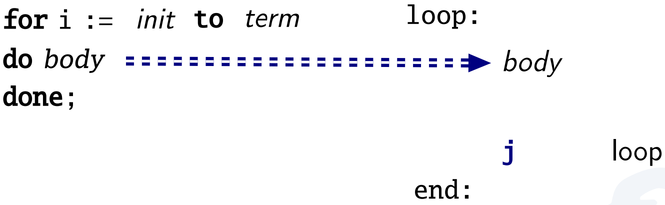


```
repeat
   body
until t0 == t1;
```

```
loop:
      body
      bne    $t0, $t1, loop
```

Other implementation options exist

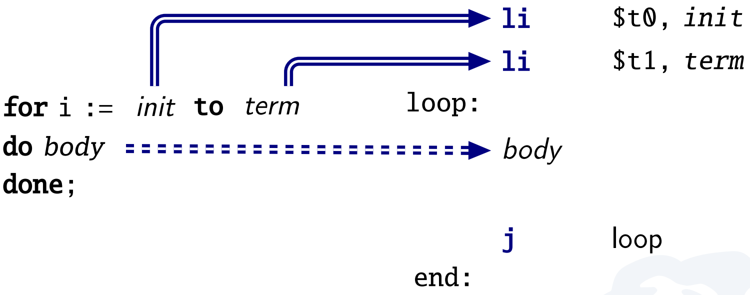
Variable-Controlled Loops



```
for i := init to term          loop:
do body  ======================>   body
done;

                                   j        loop
                               end:
```

Variable-Controlled Loops

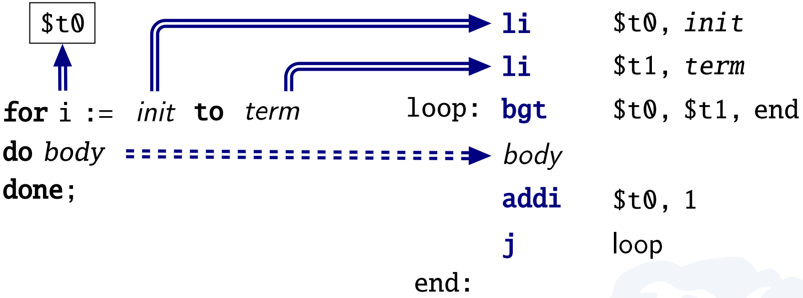


```
for i := init to term
do body
done;
```

```
        li      $t0, init
        li      $t1, term
loop:
        body


        j       loop
end:
```

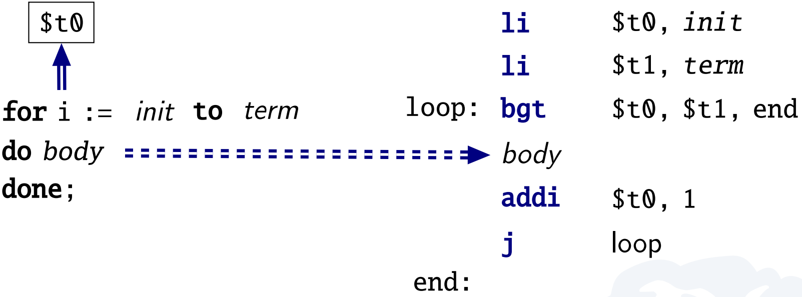
Variable-Controlled Loops



```
for i := init to term
do body
done;
```

```
        li    $t0, init
        li    $t1, term
loop:   bgt   $t0, $t1, end
        body
        addi  $t0, 1
        j     loop
end:
```

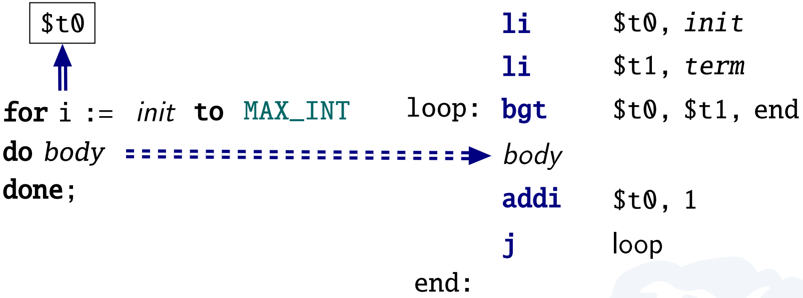
Variable-Controlled Loops



```
for i := init to term
do body
done;
```

```
       li    $t0, init
       li    $t1, term
loop:  bgt   $t0, $t1, end
       body
       addi  $t0, 1
       j     loop
end:
```

Variable-Controlled Loops



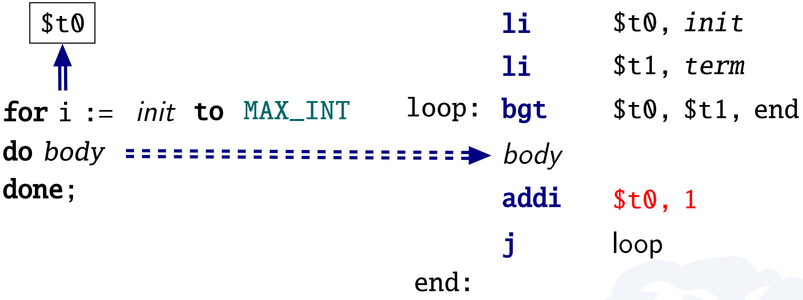
$t0

```
for i :=  init  to  MAX_INT
do body
done;
```

```
       li      $t0, init
       li      $t1, term
loop:  bgt     $t0, $t1, end
       body
       addi    $t0, 1
       j       loop
end:
```

- MAX_INT: Maximum representable integer value

Variable-Controlled Loops



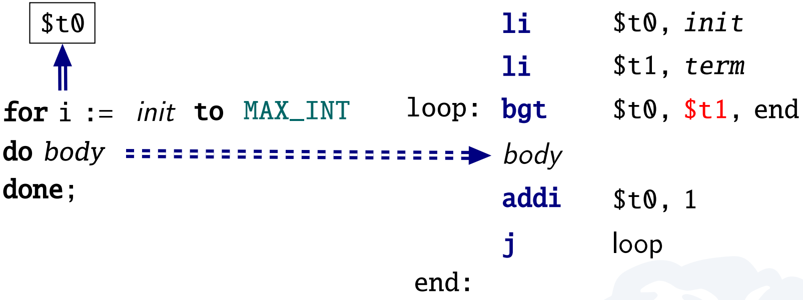
```
$t0

for i := init to MAX_INT
do body
done;
```

```
        li     $t0, init
        li     $t1, term
loop:   bgt    $t0, $t1, end
        body
        addi   $t0, 1
        j      loop
end:
```

- MAX_INT: Maximum representable integer value
- MAX_INT + 1: *overflow* in last iteration

Variable-Controlled Loops



```
$t0

for i := init to MAX_INT
do body
done;
```

```
        li    $t0, init
        li    $t1, term
loop:   bgt   $t0, $t1, end
        body
        addi  $t0, 1
        j     loop
end:
```

- MAX_INT: Maximum representable integer value
- MAX_INT + 1: *overflow* in last iteration
- If $t1 = MAX_INT, branch is never taken

Unless we know that *term* < MAX_INT, we need a different implementation strategy

Unrolling Loops: an Optimisation

```
for i := 1 to 5
do s0 := s0 * i;
done;
```

```
        li      $t0, 1
        li      $t1, 5
loop:   bgt     $t0, $t1, end
        mul     $s0, t0
        addi    $t0, 1
        j       loop
end:
```

Unrolling Loops: an Optimisation

```
for i := 1 to 5
do s0 := s0 * i;
done;
```

```
       li    $t0, 1
       li    $t1, 5
loop:  bgt   $t0, $t1, end
       mul   $s0, t0
       addi  $t0, 1
       j     loop
end:
```

```
       muli  $s0, 1
       muli  $s0, 2
       muli  $s0, 3
       muli  $s0, 4
       muli  $s0, 5
```

Only feasible if initial, terminal values and step size known

Summary

- Post-test loops:
  - Single branch
- Pre-test loops:
  - Branch before body, additional jump operation
- Variable-controlled:
  - Branch before body, additional jump operation
  - Beware: completion check nontrivial with `MAX_INT`
- Loop unrolling:
  - Optimisation when initial/terminal loop values known