TDT4205 Compiler Construction (2019)

Assignment 4

Jørgen Bele Reinfjell

March 21, 2019

1 Three Access Code and Stack Machine 10%

```
def main()
begin
    var k = gcd(1144, 546)
end

def gcd(a, b)
begin
    var g
    if b > 0 then
        g := gcd(b, a - ((a/b/)*b))
    else
        g := a
    end
    return g
end
```

1.1 TAC representation

```
main: # main()
    BeginFunc <bytes>;
    PushParam 546;
    PushParam 1144;
    k = LCall gcd;
    PopParams 16; # 2 64-bit numbers
    EndFunc;

gcd: # gcd(a, b)
    BeginFunc <bytes>;
    _t0 = 0 < b;
    IfZ _t0 Goto _L0; # if b > 0
# b is greater than 0
    _t1 = a / b;
    _t2 = _t1 * b;
    _t3 = a - _t2;
    PushParam _t3;
    PushParam b;
    g = LCall gcd # _t1 = gcd(b, a - ((a/b)*b));
    PopParams 16; # 2* 64-bits
    Goto _L1;

_L0: # else
    g = a;
_L1:
    return g;
    EndFunc;
```

1.2 Stack Layout (with assembly)

- Parameters are passed by registers (rdi, rsi, rdx, . . .).
- These parameters are saved by the caller to ensure that any subsequent calls does not overwrite them.
- Return values are returned in the rax register.

1.2.1 Layout

Stackframe offset	Identifier	Type
-8	a	param
-16	b	param
-24	g	local
-32		padding

1.2.2 When gcd(1144, 546) is about to return (inner return)

1. Stackframe #0 (gcd)

Offset	Id	Value
-8	a	1144
-16	b	546
-24	g	<unset>
-32		<unused>

2. Stackframe #1 (gcd)

Offset	Id	Value
-8	a	546
-16	b	52
-24	g	<unset>
-32		<unused>

3. Stackframe #2 (gcd)

Offset	Id	Value
-8	a	52
-16	b	26
-24	g	<unset>
-32		<unused>

4. Stackframe #3 (gcd)

Offset	Id	Value
-8	a	26
-16	b	0
-24	g	26
-32		<unused>

   - This returns 26, which is then returned by all other calls.

5. Stackframe #0 before returning to main (gcd)

Offset	Id	Value
-8	a	1144
-16	b	546
-24	g	26
-32		<unused>

1.2.3 Assembly

```
.section .rodata
    return_val: .string "gcd(%ld, %ld) = %ld\n"
.section .data
.globl main
.section .text
main:
    movq $1144, %rdi
    movq $546,  %rsi
    pushq %rdi # save
    pushq %rsi # save
    call gcd

    lea return_val(%rip), %rdi
    popq %rdx
    popq %rsi
    movq %rax, %rcx
    xorq %rax, %rax
    call printf
    ret

gcd:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq %rdi, -8(%rbp)   # param a
    movq %rsi, -16(%rbp)  # param b
    movq $0, -24(%rbp)    # local g
    movq $0, -32(%rbp)    # alignment

    #if 0 <= b then skip
    cmpq $0, -16(%rbp)
    jle skip_recurse

    # a/b
    movq $0, %rdx         # upper 64-bits
    movq -8(%rbp), %rax   # lower 64-bits
    idivq -16(%rbp)

    imulq -16(%rbp), %rax # (a/b)*b
    movq -8(%rbp), %rsi
    subq %rax, %rsi # a - (a/b)*b => %rsi

    # Recursive call to gcd(%rdi, %rsi)
    movq -16(%rbp), %rdi # b
    call gcd
    movq %rax, -24(%rbp)
    jmp done

skip_recurse:
    movq -8(%rbp), %rsi
    movq %rsi, -24(%rbp) # g := a

done:
    movq -24(%rbp), %rax # return g
    leave
    ret
```