

# Tölvutækni og forritun Heimadæmi 7

brj46

October 2024

## 1

Gefinn er smalamálskóði fyrir fallið leyndo og uppskrift að fallinu leyndo:

```
leyndo:
    cmpl    %esi, %edi
    jle     .L2
    leal     (%rdx,%rdi,2), %eax
    ret
.L2:
    leal     (%rsi,%rsi,2), %eax
    addl     %edx, %eax
    ret
```

- `cmpl` er samanburðar fall á `esi` og `edi` ( $a-b$ )
- `jle .L2` ef samanburðar fallið ef  $a$  er jafnt eða minna en  $b$  þá stökkvum við í `L2` (else hlutann)
- `leal (%rdx,%rdi,2), %eax` er reikni aðgerðin sem er gerð í `if` hlutanum  $\text{if}(a > b)$  sem reiknar  $c + 2 * a$  sem er geymd í `eax`
- `.L2` (else hlutinn)
- `leal (%rsi,%rsi,2), %eax` sem reiknar  $3 \times b$  eða  $b + 2 \times b$
- `addl edx, eax` bætir  $c$  við niðurstöðuna úr `leal` svo reikniaðgerðin í else hlutanum verður  $3 \times b + c$

```
int leyndo(int a, int b, int c)
{
    if (a > b)
        return c + 2 * a
    else
        return b * 3 + c
}
```

## 2

Við höfum fallið `abc` sem kallar á ytra fallið `xyz` og skilar summu skilagilda þeirra:

```
long xyz(long x);

long abc(long a, long b)
{
    long e = xyz(a);
    long f = xyz(b);
    return e + f;
}

abc:
    pushq    %rbp
    pushq    %rbx
    subq     $8, %rsp
    movq     %rsi, %rbp
    call     xyz
    movq     %rax, %rbx
    movq     %rbp, %rdi
    call     xyz
    addq     %rbx, %rax
    addq     $8, %rsp
    popq     %rbx
    popq     %rbp
    ret
```

### a)

Smalamálskóðinn hefur tvær `pushq` skipanir rökstyðjum tilganginn þeirra:

*pushq%rbp,%rbx*

Tilgangur þeirra er að vista `rbp` og `rbx` á hlaðanum til þess að vernda fyrri gildi skráanna til að vera notað í þessu falli. `rbp` er notað til að geyma gildi breytunnar `b`, og `rbx` er til að geyma niðurstöðuna eftir kallið á `xyz(a)`. Með því að vista þær fyrst getum við náð í þau þegar fallið er búið.

### b

Neðar í smalamálskóðanum eru þrjár `movq` skipanir rökstyðjum tilganginn þeirra.

- `movq %rsi, %rbp` færir gildi breytunnar `b` (sem er í `rsi`) yfir í `rbp` til að geyma það fyrir seinna kall.
- `movq %rax, %rbx` geymir niðurstöðuna úr `xyz(a)` í `rbx` til að viðhalda niðurstöðunni meðan við köllum á `xyz(b)`
- `movq %rbp, %rdi` færir gildi `b` sem var geymt í `rbp` í `rdi` til þess að nota það sem inntak í `xyz(b)`.

### 3

við fáum gefið fallið mitt\_switch sem inniheldur eina switch setningu.

#### a)

Hvað gerist ef tilfellunum er fækkað og hvenær hættir gcc(með -Og) að nota stökktöflu? útskýrum:

Þegar tilfellum er fækkað þá minnkar "stökktöflu", og ef tilfellum fækkar mikið þá hættir gcc að nota stökktöflu og notar í staðinn röð af samanburðum(if else skipunum).

Prófum að gera þetta með því að fjarlægja case 4 og case 5 í godbolt:

sjá á skjáskotum þar sem vinstri mynd sýnir með öllum case setningum og hægri sýnir einungis með 3 case setningum:

```
mitt_switch:
    movq    %rdx, %rcx
    cmpq    $5, %rdi
    ja      .L2
    jmp     *.L4(,%rdi,8)
.L4:
    .quad   .L2
    .quad   .L8
    .quad   .L7
    .quad   .L6
    .quad   .L5
    .quad   .L9
.L2:
    movl    $1, %eax
    ret
.L8:
    movq    %rdx, %rax
    imulq   %rsi, %rax
    ret
.L7:
    movq    %rsi, %rax
    cqto
    idivq   %rcx
    ret
.L6:
    leaq    1(%rdx), %rax
    ret
.L5:
    movl    $1, %eax
    subq    %rdx, %rax
    ret
.L9:
    movq    %rdx, %rax
    ret
```

```
mitt_switch:
    movq    %rdx, %rcx
    cmpq    $2, %rdi
    je      .L2
    cmpq    $3, %rdi
    je      .L3
    cmpq    $1, %rdi
    je      .L6
    movl    $1, %eax
    ret
.L6:
    movq    %rdx, %rax
    imulq   %rsi, %rax
    ret
.L2:
    movq    %rsi, %rax
    cqto
    idivq   %rcx
    ret
.L3:
    leaq    1(%rdx), %rax
    ret
```

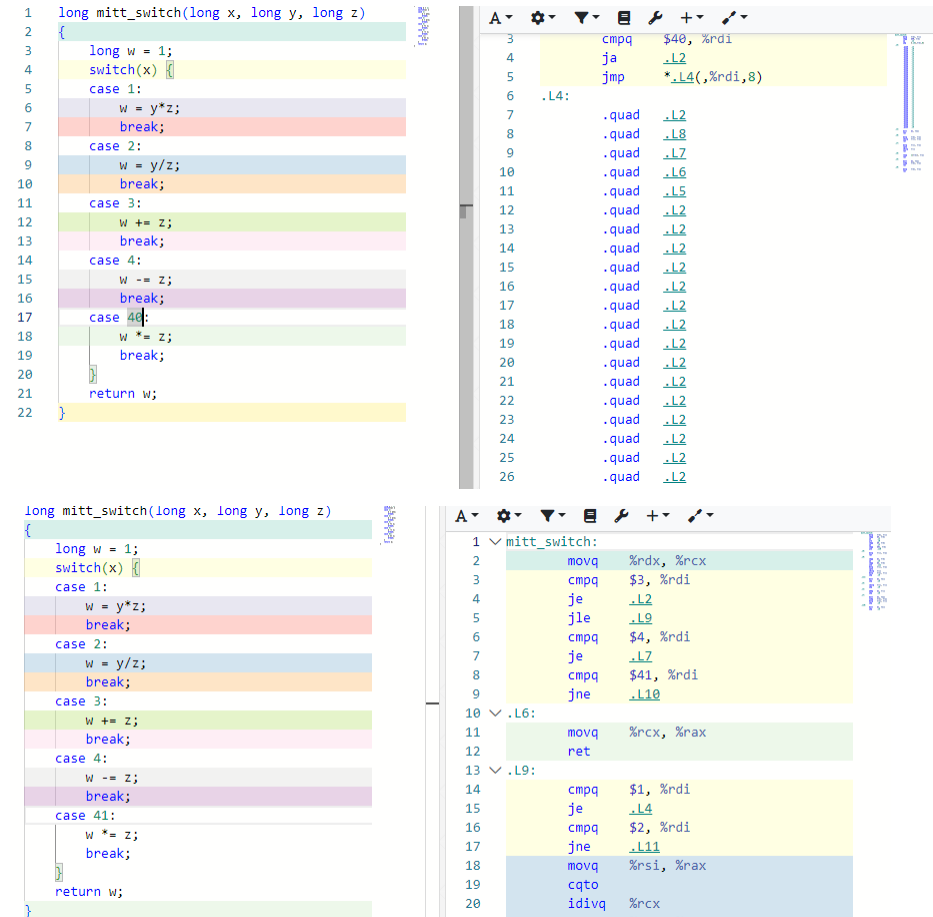
Við sjáum að gcc hættir að nota Stökktöflu og notar einungis samanburðarföll

b)

Hvað gerist ef bilið á milli tilfella er aukið?

breytum gildinu á case 5 yfir í hærri tölu þangað til að gcc hættir að nota stökktöflu:

Hjá mér hætti gcc að nota stökktöflu þegar farið var umfram case 40. sjáum á skjáskotum:



```
1 long mitt_switch(long x, long y, long z)
2 {
3     long w = 1;
4     switch(x) {
5     case 1:
6         w = y*z;
7         break;
8     case 2:
9         w = y/z;
10        break;
11    case 3:
12        w += z;
13        break;
14    case 4:
15        w -= z;
16        break;
17    case 40:
18        w *= z;
19        break;
20    }
21    return w;
22 }
```

```
3     cmpq $40, %rdi
4     ja .L2
5     jmp *.L4(,%rdi,8)
6 .L4:
7     .quad .L2
8     .quad .L8
9     .quad .L7
10    .quad .L6
11    .quad .L5
12    .quad .L2
13    .quad .L2
14    .quad .L2
15    .quad .L2
16    .quad .L2
17    .quad .L2
18    .quad .L2
19    .quad .L2
20    .quad .L2
21    .quad .L2
22    .quad .L2
23    .quad .L2
24    .quad .L2
25    .quad .L2
26    .quad .L2
```

```
1 long mitt_switch(long x, long y, long z)
2 {
3     long w = 1;
4     switch(x) {
5     case 1:
6         w = y*z;
7         break;
8     case 2:
9         w = y/z;
10        break;
11    case 3:
12        w += z;
13        break;
14    case 4:
15        w -= z;
16        break;
17    case 41:
18        w *= z;
19        break;
20    }
21    return w;
22 }
```

```
1 ~ mitt_switch:
2     movq %rdx, %rcx
3     cmpq $3, %rdi
4     je .L2
5     jle .L9
6     cmpq $4, %rdi
7     je .L7
8     cmpq $41, %rdi
9     jne .L10
10    ~ .L6:
11        movq %rcx, %rax
12        ret
13    ~ .L9:
14        cmpq $1, %rdi
15        je .L4
16        cmpq $2, %rdi
17        jne .L11
18        movq %rsi, %rax
19        cqto
20        idivq %rcx
```

c)

Hvað gerist þegar við setjum fyrstu case setninguna sem mínus tölu?:

Þegar ég breyti case 1 í case -1 birtist nír hluti í smalamálskóðanum:

*addq\$1,%rdi*

Þetta er gert til þess að nú er orðin mínus tala og með því að bæta við 1, getum við umbreytt neikvæðum gildum jákvæðar eða núlltölu svo það passi.

## 4

Við fáum gefin smalamálskóðan fun:

```
fun:
    movq    (%rdi), %rax
    jmp     .L2
.L3:
    leaq    (%rax,%rax,2), %rax
    addq    %rax, %rax
    addq    $1, %rsi
.L2:
    cmpq    %rdx, %rsi
    jl      .L3
    ret
```

### a)

Smalamálskóðinn notar þrjú gisti: %rdi, %rsi, og %rdx.

- %rdi: Fyrsta breytan er bendir á heiltölu, þar sem gildi er sótt úr minni með movq (%rdi), %rax.
- %rsi: Önnur breytan er teljari sem er aukinn með addq \$1, %rsi.
- %rdx: Þriðja breytan er efri mörk fyrir teljarann. Þetta sést í cmpq %rdx, %rsi, þar sem verið er að bera saman %rsi við %rdx. Ef %rsi er minna en %rdx, heldur lykkjan áfram.

### b)

gerum fun í c kóða:

```
long fun(long *bendir, long i, long thak){
    long result = *bendir;
    while( i < thak){
        result *= 6;
        i++;
    }
    return result;
}
```

## 5

við fáum gefinn smalamálskóða fyrir endurkvæmt fall með hausinn **int rec(int n, int m)**

```
rec:
    cmpl %esi, %edi # lína 1
    jge .L8 # lína 2
    movl $0, %eax # lína 3
    ret # lína 4
.L8:
    subq $8, %rsp # lína 5
    addl %esi, %esi # lína 6
    subl $2, %edi # lína 7
    call rec # lína 8
    addl $1, %eax # lína 9
    addq $8, %rsp # lína 10
    ret # lína 11
```

### a)

sýnum jafngilt endurkvæmt fall í C:

byrjum á að skoða hvað smalamálsfallið gerir:

- `cmpl esi,edi`: ber saman breytuna  $n$  í `edi` og  $m$  í `esi`
- `jge .L8`: Ef  $n \geq m$  fer forritið í `L8`
- `movl $0,eax`: ef  $n < m$  þá er niðurstaðan 0 sett í `eax`
- `ret` skilar niðurstöðunni 0 ef  $n < m$
- `subq $8, rsp` gerir pláss á hlaðanum.
- `addl esi, esi`: er  $m + m$  eða  $m * 2$
- `subl $2, edi`:  $n$  er minnkað um 2
- `call rec` kallar aftur á fallið `rec` með nýjum gildum
- `addl $1, eax`: bætir 1 við niðurstöðuna sem var fengið við að kalla á `rec`
- `addl $8, rsp`: tekur aftur pláss á hlaðanum sem var tekið með `subq` áður
- `ret`: skilar niðurstöðunni

Skrifum nú í C:

```
int rec(int n, int m) {
    if (n < m) {
        return 0;
    } else {
        m = 2 * m;
        n = n - 2;
        return rec(n, m) + 1;
    }
}
```

**b)**

Vísibendir (%rsp)
Færibreytur
Vistaðar skráningar
Staðværar breytur
Rammabendir(%rbp)
Endurkastfang
Hærri minnivistföng



Lægri minnivistföng