Tölvutækni og forritun Heimadæmi 7

Ragnar Björn Ingvarsson, rbi3

5. október 2024

1

Við athugum að smalamálskóðinn virkar svo að hann fyrst ber saman %esi og %edi og síðan, ef %edi er minna eða jafnt og %esi, þá hoppar forritið í .L2 en annars heldur það áfram og klárar leyndo.

Þegar það klárar leyndo setur það $\%rdx + \%rdi \cdot 2$ í %eax og skilar því gildi.

Ef það hoppar í .L2 þá setur það $\%rsi \cdot 3$ í %eax og plúsar %edx við það og skilar %eax.

Einnig er a í %rdi, b í %rsi og c í %rdx.

Svo C kóðinn er

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

2

a) Við köllum á pushq tvisvar fyrir %rbp og %rbx sem er þá til að setja upprunalegu gildi þessara gista á hlaðann til að vista þau svo að í lok fallsins getum við pop-að þeim af hlaðanum svo gildin haldist eins fyrir og eftir fallið.

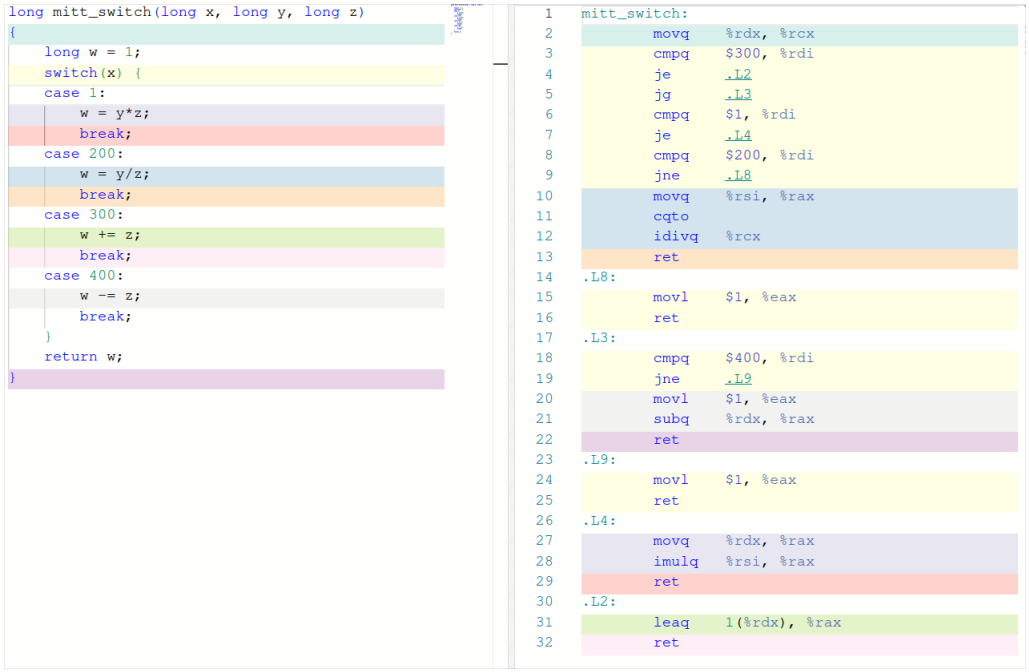
b) Movq skipanirnar þjóna mismunandi tilgangi, fyrsta er til að vista gildið b í %rbp sem leyfir okkur að vera viss um að gildið breytist ekki eftir fyrsta kallið á xyz þar sem %rbp breytist ekki fyrir og eftir köll á föll.

   Næst færum við %rax í %rbx sem er gert þar sem %rax er skilagildi fallsins xyz í fyrsta kallinu, svo við viljum geyma það skilagildi svo það yfirritist ekki af seinna kallinu.

   Síðan færum við %rbp í %rdi sem er í raun að færa vistaða gildið á b inn í %rdi sem er gistið sem notað er fyrir fyrsta inntak falls. Svo þetta er gert til að setja b sem inntak inn í seinna kall á fallið xyz.

3

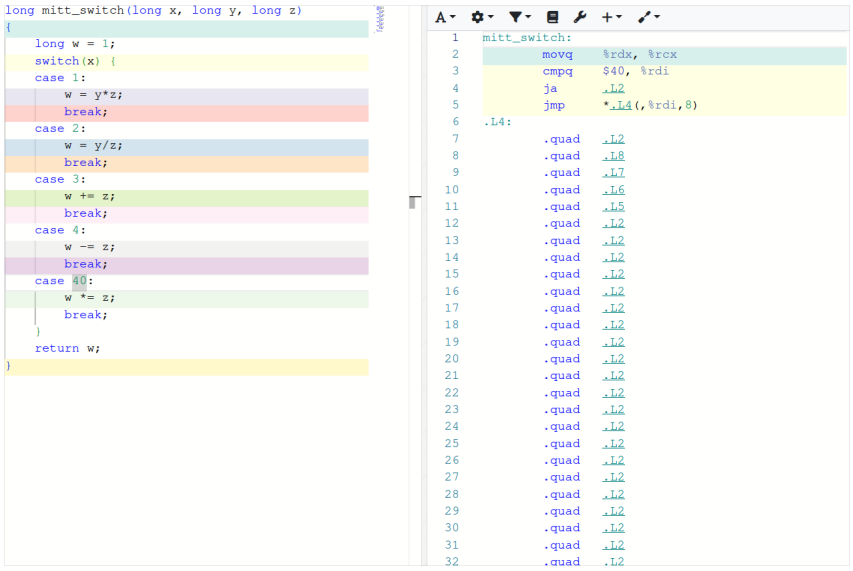
a) gcc virðist hætta að nota stökktöflu þegar við fækkum tilfellunum um einn, svo tilfellin hætta að þurfa stökktöflu þegar þau eru fækkuð niður í 4. Þetta er útkoman sama hvaða gildi við berum x við og er vegna þess væntanlega að gcc telur það hagkvæmara að framkvæma þá bara nokkrar compare skipanir í stað þess að búa til stökktöflu



```
long mitt_switch(long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 200:
        w = y/z;
        break;
    case 300:
        w += z;
        break;
    case 400:
        w -= z;
        break;
    }
    return w;
}
```

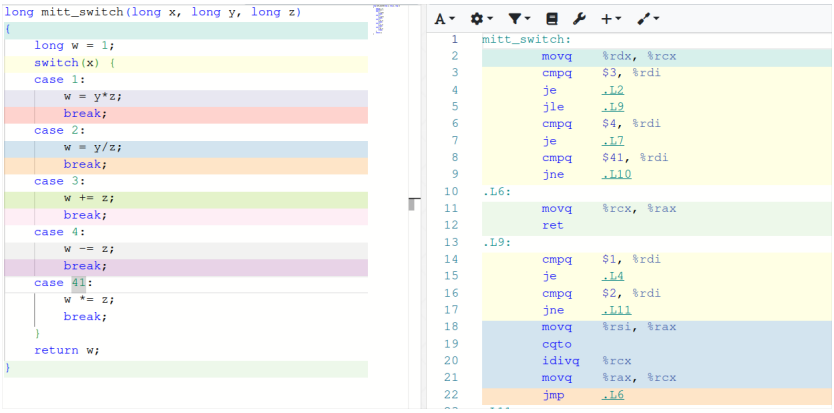
```
mitt_switch:
        movq    %rdx, %rcx
        cmpq    $300, %rdi
        je      .L2
        jg      .L3
        cmpq    $1, %rdi
        je      .L4
        cmpq    $200, %rdi
        jne     .L8
        movq    %rsi, %rax
        cqto
        idivq   %rcx
        ret
.L8:
        movl    $1, %eax
        ret
.L3:
        cmpq    $400, %rdi
        jne     .L9
        movl    $1, %eax
        subq    %rdx, %rax
        ret
.L9:
        movl    $1, %eax
        ret
.L4:
        movq    %rdx, %rax
        imulq   %rsi, %rax
        ret
.L2:
        leaq    1(%rdx), %rax
        ret
```

b) gcc hættir að nota stökktöflu fyrir öll gildi á tilfelli 5 sem eru hærri en 40, svo 40 er hæsta gildið sem gcc notar fyrir stökktöfluna.



```
long mitt_switch(long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        break;
    case 3:
        w += z;
        break;
    case 4:
        w -= z;
        break;
    case 40:
        w *= z;
        break;
    }
    return w;
}
```

```
mitt_switch:
        movq    %rdx, %rcx
        cmpq    $40, %rdi
        ja      .L2
        jmp     *.L4(,%rdi,8)
.L4:
        .quad   .L2
        .quad   .L8
        .quad   .L7
        .quad   .L6
        .quad   .L5
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
        .quad   .L2
```



```
long mitt_switch(long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        break;
    case 3:
        w += z;
        break;
    case 4:
        w -= z;
        break;
    case 41:
        w *= z;
        break;
    }
    return w;
}
```

```
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
21          movq    %rax, %rcx
22          jmp     .L6
```

c) Ef við segjum að fyrsta tilfellið byrji þá á -5, ákveður gcc að plúsa 5 við x og bera það saman við 10. Svo gcc tekur bilið sem verið er að prófa tilfelli á og ber x saman við það bil sem jákvæða tölu. Semsagt

```
addq    $5, %rdi
cmpq    $10, %rdi
```

4

a) Við sjáum að fallið notar gistin %rdi, %rsi og %rdx og skilar svo %rax. Svo fallið tekur þrjú inntök sem eru öll **long**, þar sem alltaf er notuð r útgáfan en ekki e. Einnig er fyrsta inntakið pointer þar sem náð er í gildið með (%rdi)

b) Veljum að inntökin séu a, b og c og að fallið skili x

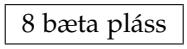
```
long fun(long *a, long b, long c)
{
    long x = *a;
    while (b < c) {
        x *= 6;
        b++;
    }
    return x;
}
```

5

a) Fallið í C lítur svona út

```
int rec(int n, int m)
{
    if (n >= m) { // comparator fyrir línur 1 og 2
        m+=m; // Lína 6
        n-=2; // Lína 7
        // Hérna sameina ég línur 8, 9 og 11, kallað er á fallið í línu 8,
        // svo er einum bætt við skilagildi fallsins í línu 9 og skilað í línu 11.
        return rec(n,m) + 1;
    }
    return 0; // Ef hoppið gerist ekki, er jafnt línum 3 og 4
}
```

b) Við sjáum að stack frameið inniheldur alltaf return addressið efst og svo tökum við 8 frá *%rsp* til að búa til 8 bæta pláss sem er sér fyrir fallið, en notum plássið ekkert í fallinu svo þetta lítur svona út:



Þar sem *%rbp* bendir á MSB og *%rsp* bendir á LSB