Confronto compilatori

1 Introduzione

In questo documento vengono confrontati dei compilatori. I Compilatori confrontati sono gcc RISC-V, gcc ARM e gcc x86_64. I compilatori RISC-V e ARM vengono confrontati per il loro focus sul ambiente embedded ed entrambi sono architetture RISC. Tutti i codici presentati in questa sezione vengono compilati e presentati con il livello di ottimizzazione di default (-O0).

- Add more Intro info

2 Operazioni

2.1 Addizione con costante

```
1 int get_num(int num) {
2     return 23 + num;
3 }
```

La funzione è una semplice funzione scritta in C che dato un numero di tipo intero restituisce il numero sommato a 23.

RISC-V

Il sorgente compilato con il compilatore RISC-V(a) da riga 2 fino a riga 6 predispone la chiamata della procedura posizionando sullo stack il necessario, da riga 7 inizia la funzione. Su quella riga viene recuperato il valore di num che alla riga 8, tramite l'operazione di add immediate, viene sommato a num. Il risultato dell'operazione addiw è la somma del valore di num sommato alla costante 23, il risultato è esteso su 64 bit, vengono ignorati gli errori di overflow. Successivamente tramite la pseudo istruzione sext.w che prende i 32 bit inferiori e li memorizza nel registro rd. Questa istruzione corrisponde a addiw rd, rs,1 0. Il risultato viene spostato nel registro a0 che, nei processori

```
addi
        s0.24(sp)
                                        push
sub
                                                 {r7}
sd
                                                 sp, sp, #12
         s0,sp,32
         a5,a0
                                        add
                                                     sp, #0
[r7, #4]
                                                                                push
mν
         a5,-20(s0)
sw
                                        str
                                                                                mov
                                        ldr
                                                                                         DWORD PTR [rbp
addiw
         a5,a5,23
                                        adds
                                                     r3, #23
                                                                                         eax, DWORD PTR
        a5, a5
sext.w
                                        mov
                                        adds
1d
        s0,24(sp)
                                        mov
                                                                                         eax. 23
                                        ldr
addi
        sp, sp, 32
                                                     [sp], #4
                                                                                         rbp
(a) RISC-V
                                          (b) ARM
                                                                                    (c) x86
```

Figure 1: Funzione di somma

RISC-V, viene utilizzato come restituzione di risultato. Le righe successive ripristinano lo stack e restituice il controllo al chiamante.

\mathbf{ARM}

l sorgente (c) , compilato con gcc ARM, mostra che la preparazione della procedure si esegue da riga 2 a riga 5, le con le due righe successive si esegue la funzione. La riga 6 recupera il valore di num la riga successiva calcola il valore del risultato e, infine, alla riga 8 si sposta il risultato nel registro di restituzione

x86

l sorgente (b) è compilato con gcc di x86. Da riga 2 fino a riga 4 viene preparato lo stack, a riga 5 viene posizionato num nel registro eax che a riga 6 viene sommato a 23 che viene memorizzato nel registro eax. Infine viene ridato il controllo al chiamante.

2.2 Addizione

Nel caso generale viene calcolata la somma di 3 numeri.

```
1 int sumGen(int num, int num2, int num3) {
2    return num + num2 + num3 ;
3 }
```

Inserisci sorgenti Addizione generale

RISC-V

Nel caso del compilatore RISC-V la somma avviene tra 13 e 19 dove gli operandi vengono caricati nei registri a4 e a5, successivamente calcolato il risultato e memorizzato in a4 che poi verrà sommato con l'ultimo operando,

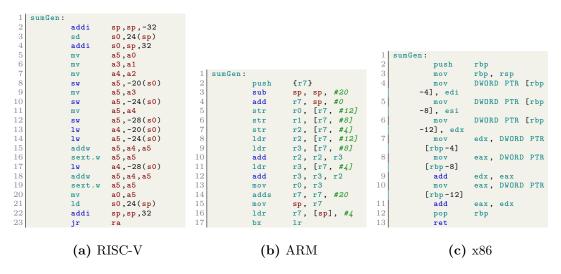


Figure 2: Funzione di somma

caricato a riga 17.

\mathbf{ARM}

Nel caso ARM avviene lo stesso meccanismo. Tra le righe 8 e 12 avviene il caricamento dei primi due operandi la somma parziale e infine la somma totale.

x86

Infine per x86 il calcolo avviene tra le righe 7 e 11 nello stesso modo con cui viene eseguito in ARM.

2.3 Moltiplicazione

Moltiplicazioni per potenze di 2

```
1 int mult2(int num){
2    return 2 * num;
3 }
```

La funzione dato un numero di tipo intero restituisce il numero moltiplicato per 2. Per i sorgenti le parti di preparazione sono simili per le rispettive preparazioni precedenti.

Nel sorgente RISC-V (2.a) l'operazione di moltiplicazione per 2 avviene tramite uno shift logical left di 1 bit (SLLIW). Stesso concetto avviene nel sor-

```
addi
                                            mult2:
                     s0.24(sp)
                                                     push
                                                               {r7}
            sd
                                                              sp, sp, #12
                                                                                     mult2:
                     s0,sp,32
            mν
                     a5,a0
                                                     add
                                                                  sp, #0
[r7, #4]
                                                                                              push
                     a5,-20(s0)
            sw
                                                     str
                                                                                              mov
                                                                                                       rbp, rsp
            lw
                      a5,-20(s0)
                                                     ldr
                                                               r3, [r7, #4]
                                                                                                       DWORD PTR [rbp
                                                              r3,
                                                              r3, r3, #1
r0, r3
            slliw
                     a5,a5,1
                                                     lsls
                                                                                           -4], edi
                                                                                                       eax, DWORD PTR
                     a5, a5
            sext.w
                                                     mov
                                                                                              mov
10
11
                      a0,a5
                                                     adds
                                                               r7, r7, #12
                                                                                             [rbp-4]
            1d
                     s0,24(sp)
                                                     mov
                                                                                              add
                                                                                                       eax. eax
12
            addi
                     sp, sp, 32
                                                     ldr
                                                                  [sp], #4
                                                                                              pop
                                                                                                       rbp
             (a) RISC-V
                                                       (b) ARM
                                                                                                  (c) x86
```

Figure 3: Moltiplicazione per 2

```
addi
             sd
                      s0,24(sp)
                                                       push
                                                                {r7}
             addi
                      s0,sp,32
                                                                sp, sp, #12
                                                                                       mult8:
                                                                r7, sp, #0
r0, [r7, #4]
r3, [r7, #4]
                      a5,a0
                                                       add
                                                                                                 push
                      a5,-20(s0)
             sw
                                                       str
                                                                                                 mov
                      a5,-20(s0)
                                                                                                 mov
                                                                                                          DWORD PTR [rbp
             slliw
                      a5,a5,3
                                                       lsls
                                                                r3, r3, #3
                                                                                              -4], edi
                                                                                                          eax, DWORD PTR
             sext.w
                      a5, a5
                                                       mov
                                                                r0, r3
11
             1d
                      s0.24(sp)
                                                       mov
                                                                                                 sal
                                                                                                          eax. 3
12
                                                       ldr
             addi
                                                                r7, [sp], #4
                      sp, sp, 32
                                                                                                 pop
                                                                                                          rbp
             (a) RISC-V
                                                         (b) ARM
                                                                                                    (c) x86
```

Figure 4: Moltiplicazione per 8

gente ARM (2.c) dove l'operazione di moltiplicazione per 2 avviene tramite lo shift left di 1 bit. Invece nel sorgente x86(2.b) la moltiplicazione avviene tramite una somma. Questa somma è un caso particolare, infatti se volessimo moltiplicare per una qualsiasi potenza di 2 (ad esempio in figura 3) le operazioni avvengono tutte tramite shift left di un opportuno valore. Con la figura 4 viene mostrato il calcolo di una moltiplicazione per 8. Nei casi RISC-V e ARM il calcolo avviene attraverso Shift e nel caso x86 avviene anche qui uno shift.

Moltiplicazione per una costante

Vengono presentati due codici molto simili, il primo moltiplica il numero per 31 che rappresenta più in generale un numero che dista da una potenza di 2 di 1. Il secondo è un caso più generale dove avviene la moltiplicazione di un numero non potenza di due e che dista da una potenza almeno di 2, nel nostro caso il numero è 30.

```
int mul31(int a) {
        return a * 31;
                                                                              return a * 30;
             addi
                      sp,sp,-32
s0,24(sp)
                                                      push
                                                               {r7}
             sd
             addi
                                                               sp, sp, #12
                                                                                      mul31:
                      s0,sp,32
                                                                   sp, #0
[r7, #4]
             mν
                      a5,a0
                                                      add
                                                                                               push
                      a5,-20(s0)
             sw
                                                      str
                                                                                               mov
                                                                                                        rbp, rsp
                                                                                                         DWORD PTR [rbp
             mν
                      a5.a4
                                                      mov
                                                               r3, r2
                                                                                            -4], edi
             slliw
                      a5,a5,5
                                                      lsls
                                                               r3, r3, #5
                                                                                                         edx, DWORD PTR
                                                                                               mov
10
11
12
             sext.w
                      a5.a5
                                                      mov
                                                               r0, r3
                                                                                               mov
                                                                                                        eax. edx
                      a0,a5
                                                      adds
                                                               r7, r7, #12
             mν
                                                                                               sal
                                                                                                         eax, 5
13
14
15
             1d
                      s0,24(sp)
                                                                                                         eax, edx
                                                               r7, [sp], #4
             addi
                      sp,sp,32
                                                      ldr
                                                                                                        rbp
             (a) RISC-V
                                                        (b) ARM
                                                                                                   (c) x86
```

Figure 6: Moltiplicazione per 31

Nel caso della moltiplicazione per 31 l'approccio dei 3 sorgenti è identico. Viene calcolata la moltiplicazione per 32 attraverso shift logici e poi viene sottratto una volta il valore per ottenere la moltiplicazione per 31. Se il valore costante fosse 33, il numero successivo alla potenza di 2, l'operazione di sottrazione viene sostituita con una di addizione. Nel caso più generale invece abbiamo un approccio differente.

Partendo dall x86 la moltiplicazione avviene semplicemente con l'istruzione imul a riga 6. Nel caso ARM l'operazione di moltiplicazione viene comunque eseguita da una singola istruzione(riga 8) ma vengono utilizzati i registri r2 e r3 che precedentemente (riga 6 e 7) vengono riempiti con gli operandi. Infine l' implementazione di RISC-V utilizza ancora shift. Nel caso della moltiplicazione per 30 avviene prima uno shift di 4 (moltiplicazione per 16) successivamente sottratto una volta il numero e infine al risultato avviene applicato uno shift di 1 (moltiplicazione per 2).

Quindi:

```
((num * 2^4) - num) * 2^1 =
= ((num * 16) - num) * 2 =
= 15 * num * 2 = num * 30
```

In generale RISC-V utilizza opportuni shift combinate con addizioni e sottrazioni per ottenere il valore della costante.

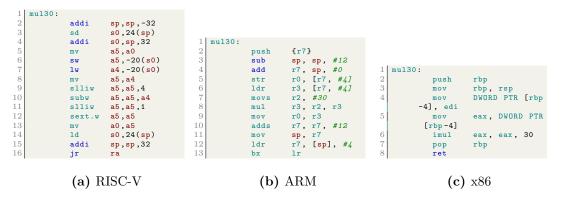


Figure 7: Moltiplicazione per 30

2.4 Divisione

Per quanto riguarda la divisione viene utilizzato la stessa metodologia della moltiplicazione.