GoTalks - 20. 06. 2023. - Zagreb

Go ASM - deep dive

sartura

Go ASM

- Assembly jezik, assembly ili ASM/asm je low-level programski jezik koji je prvi korak iznad strojnog jezika, najčešće nečitljiv ljudima (iako bi mnogi rekli da je i asm takav :))
- asm koristi mnemonike za reprezentaciju svake low-level strojne instrukcije ili opkoda
- Assembler je softver koji kreira objektni kod putem određene sintakse i prevođenja mnemonika
- Svaka arhitektura ima svoj assembly jezik

Go ASM

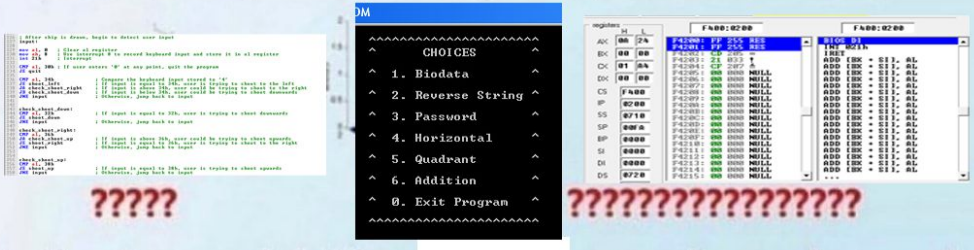
- Tko to danas koristi?
- Za “generalno” korištenje, potpuno besmisleno (entuzijasti poput Chrisa Sawyera koji je napisao RollerCoaster Tycoon isključeni)
- Za pristup “naprednom” hardveru - nužnost

STOP DOING ASSEMBLY

- ROCKS WERE NOT SUPPOSED TO BE GIVEN NAMES
- YEARS OF ASSEMBLY yet NO REAL-WORLD USE FOUND for creating anything but PACMAN
- Wanted to write some code anyway? We had a tool for that: It was called "BINARY"
- "Yes please mov ax, 0C02h. Please CMP al, 38h" - Statements dreamed up by the utterly Deranged

LOOK at what PROGRAMMERS have been demanding your Respect for all this time, with all the higher level languages we built for them

(This is REAL ASSEMBLY done by REAL PROGRAMMERS):



?????

??????????????????

"Hello I would like 2BE9F apples please"

They have played us for absolute fools

Go ASM

- Što smatramo “naprednim hardverom”?
- U kontekstu našeg razmatranja, SIMD je glavni razlog
- Već gotovo 10 godina, standardni x64 C compileri koriste SSE instrukcije za matematičke operacije
- Razlog tome je umirovljenje x87 instrukcija, doduše (AMD ih već godinama ne implementira, MSVC kompajler ih eksplicitno ne dozvoljava na x64 buildovima, itd.)
- No, SIMD je dogurao daleko, pa danas imamo puninu CISC kompleksnosti operacija
- Razlog za korištenjem SIMD instrukcija može se pokazati na vrlo “pitkim” primjerima, no mi ćemo za jedan od primjera odabrati nešto daleko prozaičnije

Go ASM

- Upoznajte - PCMPxSTRx
- Radi se o 4 instrukcije:
  - `PCMPESTRI` - Explicit length, return index
  - `PCMPESTRM` - Explicit length, return mask
  - `PCMPISTRI` - Implicit length, return index
  - `PCMPISTRM` - Implicit length, return mask
- Osnovni cilj - razne operacije uspoređivanja i pretraživanja stringova

Go ASM

- Explicit - duljina je specificirana u RAX i RDX registrima
- Implicitno - stringovi su NULL ili \0 terminirani
- Maska - rezultati se vraćaju kao 0/1 maska
- Index - indeks prvog ili zadnjeg rezultata je vraćen u RCX
- Tri operanda - xmm, xmm/mem, imm8
  - xmm - uzorak koji se traži/poklapa
  - xmm/mem - string kojeg pretražujemo
  - imm8 - bitmaska

Go ASM

- imm8 - 00 00 00 00b
  - 00 - unsigned bytes
  - 01 - unsigned words
  - 10 - signed bytes
  - 11 - signed words
  - 00 - podskup (pronađi bilo koji znak specificiran u uzorku bez obzira na poredak:

    ```
    string to search
    pattern
    ```

Go ASM

- 01 - pronađi znakove specificirane po dometu parova, primjerice AZ, 09 i slično:

  F1**nD** 7**h**3 **L**3773**r**5

  AZaz
- 10 - poklapanje:

  AT**T**CAT**G**TTC**C**C**TG**CA

  GCTAGGGACACATGTT

Go ASM

- 11 - traži podskup:
  ABBABA**ABBA**BAAB**AB**
  ABBA
  Napomena: ovisno o kodiranju duljina stringa, zadnji "AB" može ili ne mora biti u skupu pronađenih podskupova
- 11 - LSB komplementira rezultat a MSB primjenjuje komplement samo na validne bitove
- Ukoliko je bila odabrana maska - 0 vraća rezultat kao bit masku, 1 vraća rezultat kao byte masku; inače, 0 vraća indeks prvog poklapanja u RCX, odnosno 1 vraća indeks zadnjeg poklapanja u RCX
- mora biti 0

Go ASM

- Ima i ljepših instrukcija (nota bene, prethodno navedenu je ubio i sam Intel neuvevši String and Text New Instructions (STNI) u AVX2 instrukcije i dramatično produžio izvršavanje ove instrukcije na Haswell+ generaciji procesora na 11-12 ciklusa, što ih čini praktički neupotrebljivima)
- Obična aritmetika je najjedostavniji primjer
- Klasične skalarne operacije izgledaju ovako:
  - `1 + 2 = 3`
  - `3 + 4 = 7`
  - ...
  - `an + bn = cn`

Go ASM

- Ukoliko ovakve operacije izvršavamo na jednom procesoru/jezgri, to zahtijeva: *n* čitanja iz *a*, *n* čitanja iz *b*, *n* operacija zbrajanja, *n* zapisivanja u *c*
- Ukupno – *4 x n* operacija

Go ASM

- SIMD operacije nude puno bolju alternativu
- 1 2 3
- 3 4 7
- . . .
- \+ =
- . . .
- an bn cn

- n vrijednosti učita se iz a u jednoj operaciji
- n vrijednosti učitava se iz b u jednoj operaciji
- n zbrajanja u jednoj operaciji
- n zapisivanja u jednoj operaciji
- Ukupno: 4 operacije
- Fantastično za matrične račune, kriptografiju i slično
- Motivacija: koristiti takve instrukcije u Gou!

Go ASM

- Go kompajler ne generira automatski većinu SIMD instrukcija
- Kako stvari ne bi bile jednostavne, čak i sama službena dokumentacija (https://go.dev/doc/asm) kaže:
  - This document is a quick outline of the **unusual form** of assembly language used by the gc Go compiler. **The document is not comprehensive.** (*op.a. ima se što za čitati satak barem*)
- No, uvedimo prije svega nekoliko pojmova
- Mi ćemo, kako je to praksa u Gou, kompajlirati naš assembly iz C-a
- Razlog tome je jednostavnost - C podržava assembly intrinzike koji uvelike olakšavaju pisanje koda

Go ASM

- Implementirat ćemo VSQRTPS mnemonik - Square Root of Single-Precision Floating-Point Values
- Koristit ćemo *_mm512_sqrt_ps* intrinzik iz immintrin.h zaglavlja
- Source: https://pastebin.com/vFjpACse

Go ASM

- *clang -S -c mm512_mul_to.c -o mm512_mul_to.s -03 -mavx -mfma -mavx512f -mavx512dq -mno-red-zone -mstackrealign -mllvm -inline-threshold=1000 -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti*
- *clang -c mm512_mul_to.c -o mm512_mul_to.o -03 -mavx -mfma -mavx512f -mavx512dq -mno-red-zone -mstackrealign -mllvm -inline-threshold=1000 -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti*

Go ASM

- Koristeći objdump, dobit ćemo strojni jezik iz našeg binaryja
- `objdump -d mm512_mul_to.o --insn-width 16`
- Sjećate se čudne assembly sintakse za Go? Postoje tri instrukcije za reprezentaciju binarnog strojnog jezika
  - BYTE - kodira jedan byte binarnog podatka
  - WORD - dva
  - LONG - četiri

Go ASM

- Ukoliko je instrukcija strojnog jezika duljine djeljive s dva, primjerice:

```
50:    62 d1 7c 48 51 06 vsqrtps (%r14),%zmm0
```

Pretvaramo je u:

```
LONG $0x487cd162; WORD $0x0651 // vsqrtps (%r14),%zmm0
```

- Primjetite da je poredak bajtova obrnut

Go ASM

- Ako duljina nije djeljiva s 2:

  ```
  56: 62 d1 7c 48 11 45 00 vmovups %zmm0,0x0(%r13)
  ```

  Potrebna je kombinacija više instrukcija za prevođenje:

  ```
  LONG $0x487cd162; WORD $0x4511; BYTE $0x00 //
  vmovups    %zmm0, (%r13)
  ```

- Također, još jedna od specifičnosti je sljedeća - ako u C assembliju funkcija nema više od 6 argumenata, argumenti su pohranjeni u registre i prosljeđuju se u funkciju po sljedećem poretku: %rdi, %rsi, %rdx, %rcx, %r8, %r9 (P.S. - ako ih je više, idu na stog; više o toj temi možete naći pod pojmom “calling convention”)

Go ASM

- Go pak drži funkcijske argumente u memoriji počevši od adrese u FP registru, pa prije svega moramo pomaknuti argumente iz memorije u registre
- S obzirom da naša *_mm512_sqrt* funkcija prima tri argumenta, potreban nam je sljedeći translacijski dio:

```
TEXT ·_mm512_sqrt(SB), $0-32

     MOVQ a+0(FP), DI

     MOVQ b+8(FP), SI

     MOVQ n+16(FP), DX
```

Go ASM

- Funkcijska definicija sadrži 3 dijela: ključnu riječ TEXT, ime koje počinje sa “–” i završava sa “(SB”), te parametra memorijske veličine od 32 bajta
- Informacija o argumentima nije dostupna assembliju te ju je potrebno dohvatiti direktno iz definicije C funkcije
- Potrebne su još dvije sitnice za naš konačni proizvod - assembly tagovi u Gou ne smiju počinjati s točkama, pa ih treba maknuti, te jump naredbe moraju biti kapitalizirane (jmp vs JMP)

Go ASM

- Konačno, potrebno je napraviti Go funkcijsku definiciju
- Ime je ekvivalentno C funkciji
- Argumenti su prosljeđeni kao unsafe.Pointer

```
package main
import "unsafe"
//go:noescape
func _mm512_sqrt(a, b, n unsafe.Pointer)
```

- Sve ovo možete napraviti rukom, ukoliko vam standardni skup pokora nije adekvatan

Go ASM

- Ipak, postoji dobra alternativa: https://github.com/gorse-io/gorse/tree/master/cmd/goat
- Kako ja znam da je ta muka stvarno brža?
- https://pastebin.com/vaMBWBLe
- go test -bench=. -benchtime=5s -count=5

Hvala na pažnji!

Bruno Banelli – bruno.banelli@sartura.hr