Laboratorul 7.
Arhitectura MIPS

Hello world

Pornind de la codul de la curs, scrieți un program in assembly de MIPS care afișează "Hello World". Consultați paginile A-48 - A-49 din manualul de SPIM[1].

Pentru a putea încărca codul în SPIM[2], trebuie să faceți următoarele modificări:

- Adăugați "entry-point"-ul (locația din zona de cod de unde se va începe execuția programului). Acesta este indicat printr-o etichetă de `main:` .
- Indicați finalul programului. Dacă acesta nu este specificat, contorul de program va fi incrementat la nesfârșit (până la întâlnirea unei erori sau a unei instrucțiuni invalide). Finalul este indicat de `syscall`-ul pentru `exit`:

```
li $v0, 10 #exit
syscall
```

  Folosiți butoanele "Run/Continue" și "Clear registers" pentru a rula codul de oricâte ori. Asigurați-vă că fereastra de `Console` este vizibilă.

If-Then

Traduceți din pseudocod în assembly următorul fragment:

```
print_string("1");
if N > 64 then
       print_string("Large value")
end-if
print_string("2");
```

If-Then-Else

Traduceți din pseudocod în assembly următorul fragment:

```
print_string("1");
if N > 64 then
       print_string("Large value")
else
       print_string("Small value")
end-if
print_string("2");
```

[1]Disponibil aici: `https://curs.upb.ro/mod/resource/view.php?id=42337`
[2]Installer disponibil aici: `https://sourceforge.net/projects/spimsimulator/`

While loop

Afișați pe consolă numerele între 0 și 20, în ordine crescătoare. Acesta este echivalentul unui while:

```
unsigned int i = 0;
while(i <= 20)
       print_int(i)
       i++
```

Stack-pointer

Încărcați pe stivă numerele între 0 și 20. Apoi, printați-le folosind o nouă buclă, ce scoate numerele de pe stivă în timp ce le afișează.

Notă: Stiva "crește" spre adresele joase și "scade" către adresele mari.

Fizzbuzz

Rezolvați problema "Fizzbuzz" în assembly:

```
unsigned int i = 0;
while(i <= 100)
       if (i mod 15 == 0)
              print_string("Fizzbuzz")
       else if (i mod 3 == 0)
              print_string("Fizz")
       else if (i mod 5 == 0)
              print_string("Buzz")
       else
              print_int(i)
       i++
```

Notă: Restul împărțirii este calculat de operația `divu` și stocat în registrul `hi`. Copiați restul folosind `mfhi`.

Bonus: Floating point numbers

Definiți un array de valori floating-point. Folosind instrucțiunile de FPU (`lwc1`, `cvt.w.s`, `c.le.s`, `bc1t`, `mfc1`), afișati valoarea maximă din acel array și conversia acestuia în Int.

Bonus: Interpretarea output-ului de GCC

Folosind site-ul "Godbolt Compiler Explorer" (`https://godbolt.org/`) urmăriți instrucțiunile de MIPS generate pentru un program simplu scris în C.

Bonus: Asamblarea și rularea în QEMU (Linux-only)

Până acum ați folosit emulatorul QtSpim, care pentru operațiile de I/O folosește interfața proprie bazată pe `syscall` (fiind echivalentul unui firmware/BIOS propriu). Pentru exercițiul următor vom folosi emulatorul QEMU pentru a executa binare de MIPS pe calculatoare cu arhitecturi Intel, folosind interfața de `syscall` de Linux.

Convenția de apel pentru Linux este următoarea:

- `v0` - Codul operației de `syscall`. Pentru MIPS este documentat aici: `http://git.linux-mips.org/cgit/ralf/linux.git/tree/arch/mips/include/uapi/asm/unistd.h`. Documentația privind argumentele primite de fiecare `syscall` este aici: `https://syscalls.kernelgrok.com/`.
- `a0` - Primul parametru;
- `a1` - Al doilea parametru;
- `a2` - Al treilea parametru.

Transcrieți următorul program:

```
.globl __start
.data

# store the string
buffer:
.asciiz "hello world\n"

# store the string length
buffer_len:
.word 12

.text
__start:
# write(stdout, buffer, buffer_len)
li $v0, 4004 # 4004 - syscall code for "write"
li $a0, 1 # write to STDOUT (file descriptor 1)
# load $a1 register in two parts:
# first the high 16 bits, then the low ones
lui $a1, %hi(buffer)
addiu $a1, $a1, %lo(buffer)
lw $a2, buffer_len # specify the length of the string
syscall # run the syscall

# exit(0)
li $v0, 4001 # 4001 - syscall code for "exit"
li $a0, 0 # return value (success = 0)
syscall # run the syscall
```

Instalați programele necesare:

```
sudo apt-get install binutils-mips-linux-gnu qemu-user \
qemu-system-mips qemu-user-binfmt
```

După care urmează obținerea executabilului de MIPS:

```
$ mips-linux-gnu-as hello.as # generam fisierul de cod-obiect
$ mips-linux-gnu-ld a.out -o bin # transformam codul-obiect in executabil
$ file bin # verificare
bin: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically
linked, not stripped
```

Putem încărca executabilul în QEMU, folosit ca "user-mode emulator". Asta înseamnă ca tot contextul aplicației va fi emulat, iar `syscall`-urile (tranzițiile în kernel-space) vor fi transformate on-the-fly în apeluri native de x86 (și vor fi tratate de kernel-ul de pe host).

```
$ qemu-mips bin
hello world
```

Bonus: Binfmt_misc (Linux-only)

Încercați să rulați direct binarul generat pentru arhitectura MIPS.

```
$ file bin
bin: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically
linked, not stripped

$ ./bin
hello world
```

Dacă s-a executat transparent, se datorează extensiei de binfmt_misc. Sistemul Linux detectează că binarul este specific altei arhitecturi, și pornește automat o instanță de `qemu-mips` în care încarcă acel binar.