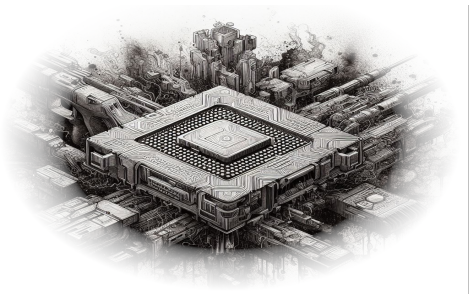
Úloha č. 3
Bitva v paměti



Odpověz sfinze! 10 b

Tato úloha je vyhodnocována automaticky. Je potřeba, aby výstup programu ***přesně*** *korespondoval se specifikací výstupu níže. Jak odevzdávat tento typ úloh se můžeš dočíst na webových stránkách FIKSu pod záložkou „Jak řešit FIKS“.*

Se seznamem všech potřebných spojení jsme se dali do práce. Cestovali z uzlu do uzlu a vytvářeli vše potřebné. S Gimlim a Gandalfem jsme v jednom uzlu čekali na ostatní. Konečně se před námi vytvořila černá, logicky bílá díra.

„No konečně, že jste tady,“ zvolal Gandalf. Místo Aragorna a Legolase ale z černé, logicky bílé díry vyskočila vyhublá postava v šedém děravém plášti. Pod pláštěm bylo staré triko s nějakým kódem, z pod kterého lezly žebra. Některá ovšem chyběla a byla nahrazena žebry procesorového chladiče. Ruce volně splývaly podél těla a místo některých prstů byly šroubováky, kleště a drát, který připomínal oko pájky. V oblasti hlavy byla velká díra, neboť místo pravé mozkové hemisféry byla zasazena patice k procesoru. Levé oko vypadalo spíše jako rozbitá čočka kamery a místo pusy byla disketová mechanika, ze které koukaly zbytky řízku.

Gandalf pozvedl svoji hůl a mocným zmáčknutím tlačítka vyslal k postavě magnetický impulz. Postava se podlomila v kolenech a s rachotem spadla na zem a roztříštila se. Z jednotlivých úlomků se rozeběhly bugy – malé kusy PCB, které vypadaly jak části všemožných konektorů. Hlavně ISA, PCI a USB. Bugy se rozutekly po místnosti a vložily se do nepoužitých portů vší elektroniky, která se v místnosti nacházela.

„Rychle, nesmí se dostat k paměti,“ pronesl klidně Gandalf. Bylo bohužel už pozdě. Když se podíval do paměti uzlu, byla plná náhodného kódu, který nešel vymazat. „Je potřeba vytvořit vlastní, který náhodný kód odstraní.“

Tvým cílem bude napsat program, který je nejen schopen přežít, ale i zničit ostatní programy – budeš tedy bojovat jak s předpřipravenými programy, tak s programy svých kolegů. Nebude to ale tak jednoduché, jak by se mohlo zdát. Tvůj program bude běžet na počítači, který má několik specifických vlastností.

Narozdíl od běžného programování tu je pár drobných rozdílů.

- Paměť pro data a kód je společná. To znamená, že můžeš přepsat svůj vlastní kód. Stačí zapsat do paměti na správné místo správné byty.
- Na počítači běží několik (až 8) procesů, ty ale mají společnou paměť. To znamená, že můžeš přepsat kód jiného procesu pokud zjistíš, kde běží.
- Paměť je omezená. Tvůj kód se navíc musí vejít do 256 instrukcí.
- Kromě paměti (která je sdílená pro všechny programy) má každý program své vlastní registry. Registr je taková proměnná, která je ale čistě pro tvůj program a nikdo jiný ji nemůže číst ani do ní zapisovat. Ve všech registrech je na začátku nula. V prvním registru (R0) je vždy nula a

není povolené do něj zapisovat. Registry nemají jména, ale čísla, indexuje se od nuly. Registrů je 6 a mají velikost 32 bitů.

- Chování při přetečení (tedy pokusu uložit větší číslo, než se vejde) jak paměti tak registrů není definované – může se stát cokoliv.

Tvůj program běží, dokud nenastane některé z následujícího:

- Vykonal jsi neplatnou instrukci. Tedy něco, co není v tabulce níže. Toto se může snadno stát, pokud jsi neopatrně zapisoval do paměti a špatně sis přepsal instrukce.
- Do neplatných instrukcí spadá i přístup mimo rozsah paměti.
- Vykonal jsi instrukci „bomba“ ve chvíli, kdy zbývala na odpočtu nula.
- Zacyklil jsi se.

Tvým cílem je běžet déle, než programy ostatních. (Tedy jednou z možných metod přežití může být zkusit zničit ostatní programy.)

Registry

Číslo	Popis
0	Vždy obsahuje nulu. Do něj nelze zapisovat.
1	Druhý registr.
2	Třetí registr.
3	Čtvrtý registr.
4	Pátý registr.
5	Šestý registr.

Paměť

Paměť má předem neznámý rozsah. Lze ji indexovat klasicky jako u normálního pole, tedy lze číst například z `mem[10]` nebo zapisovat na `mem[11]`.

- Vždy obsahuje na začátku $256 * 4$ bytů, které „nepatří“ žádnému hráči. Najdeš tam také speciální adresy, ze kterých jde jenom číst:

Adresa	Popis
42 (0x2a)	Obsahuje adresu nejbližšího programu[1].
43 (0x2b)	Obsahuje adresu druhého nejbližšího programu.

- Poté pro každého hráče obsahuje dalších $256 * 4$ bytů, kde se nachází jeho počáteční program.
- Nevíš, v jakém pořadí jsi ani kolik programů běží - tedy jestli jsou před tebou nějaké programy, nebo ne.

[1] Počítáno od místa, kde zrovna vykonáváš kód.

Instrukce

Každá instrukce je 4 byty (32 bitů) dlouhá. První byte je kód instrukce, další tři jsou argumenty.

Všechna čísla ve slouepčku „Kód“ uvedená v této tabulce jsou hexadecimální. Tedy například číslo 69 je ve skutečnosti `0x69`, což je v desítkové soustavě 105. Dvojice čísel (v tomto případě tedy třeba 69 je jeden byte = 8 bitů.)

Ve sloupečku kód je předpis instrukce. Když je někde číslo, je pevně dané. Otazníky znamená, že hodnota daných bitů může být libovolná a nezáleží na ní. Zápis typu `reg1(4b)` znamená, že následují 4 bity, které určují registr, který instrukce použije. Zápis typu `imm(16b)` znamená, že následuje 16 bitů, které určují číslo (vyloženě číselný argument), které instrukce použije.

Výrazem `pc` se myslí adresa právě vykonávané instrukce. Tato hodnota se inkrementuje po každém vykonání instrukce, pokud neproběhla speciální instrukce, která hodnotu `pc` mění - například skoky.

Neboj se, pokud to zatím nedává smysl, z příkladů bude vše jasné.

Kód	Instrukce	Popis	Příklad
`69 ??(8b) ??(8b) ??(8b)`	`NOP`	Nic neudělá.	`69 de ad be`
`01 reg1(4b) reg2(4b) imm(16b)`	`ADD`	`reg1 += reg2 + imm`	`01 12 00 04` - k hodnotě prvního registru přičte hodnotu registru 2 plus 4 a výsledek uloží do prvního registru.
`02 reg1(4b) reg2(4b) imm(16b)`	`SUB`	`reg1 -= reg2 + imm`	`02 34 00 A4` - od hodnoty třetího registru odečte (hodnotu registru 4 plus `0xA4` (tedy 164)) a výsledek uloží do třetího registru.
`03 reg1(4b) reg2(4b) imm(16b)`	`MUL`	`reg1 *= reg2 + imm`	`03 50 FF EE` - hodnotu pátého registru vynásobí (hodnotou registru 0 plus `0xFFEE`) a výsledek uloží do páteho registru.
`05 reg1(4b) reg2(4b) imm(16b)`	`LOAD`	`reg1 = mem[reg2+imm]`	`05 13 00 00` - hodnotu prvního registru nastaví na hodnotu v paměti na adrese (hodnota třetího registru plus 0).
`06 reg1(4b) reg2(4b) imm(16b)`	`STORE`	`mem[reg2+imm] = reg1`	`06 04 00 0A` - hodnotu v paměti na adrese (hodnota čtvrtého registru plus 10) nastaví na hodnotu nultého registru (tedy zapíše nulu).
`07 reg1(4b) reg2(4b) imm(16b)`	`MOV`	`reg1 = reg2 + imm`	`07 12 01 28` - hodnotu prvního registru nastaví na hodnotu druhého registru plus `0x128`.
`10 reg1(4b) reg2(4b) imm(16b)`	`JUMP`	`if (reg1 == reg2) pc += imm`	`10 12 00 04` - pokud je hodnota prvního registru rovna hodnotě druhého registru, skočí o čtyři instrukce dopředu.
`11 reg1(4b) reg2(4b) imm(16b)`	`REVJUMP`	`if (reg1 == reg2) pc -= imm`	`11 00 00 01` - pokud je hodnota nultého registru rovna hodnotě nultého registru, skočí o 1 instrukci dozadu.

12 reg1(4b) reg2(4b) imm(16b)	LTJUMP	if (reg1 < reg2) pc += imm	12 12 00 04 - pokud je hodnota prvního registru menší než hodnota druhého registru, skočí o čtyři instrukce dopředu.
13 reg1(4b) reg2(4b) imm(16b)	REVLTJUMP	if (reg1 < reg2) pc -= imm	13 50 00 08 - pokud je hodnota páteho registru menší než hodnota nultého registru, skočí o osm instrukcí dozadu.
14 reg1(4b) reg2(4b) imm(16b)	NEQJUMP	if (reg1 != reg2) pc += imm	14 12 0A 00 - pokud je hodnota prvního registru nerovna hodnotě druhého registru, skočí o 0xA00 instrukce dopředu.
15 reg1(4b) reg2(4b) imm(16b)	REVNEQJUMP	if (reg1 != reg2) pc -= imm	15 12 00 04 - pokud je hodnota prvního registru nerovna hodnotě druhého registru, skočí o čtyři instrukce dozadu.
20 reg1(4b) ??(4b) imm(16b)	SETIMMLOW	reg1[low] = imm	20 1F 00 04 - nastaví dolních 16 bitů prvního registru na hodnotu 4.
21 reg1(4b) ??(4b) imm(16b)	SETIMMHIGH	reg1[high] = imm	21 1C 00 04 - nastaví horních 16 bitů prvního registru na hodnotu 4.
42 imm1(8b) imm2(16b)	TELEPORT	Zmrazí program. Pokud jiný program také spustí instrukci TELEPORT, tyto dva programy si okamžitě prohodí místa. Pokud žádný program TELEPORT nezavolá po dobu imm1 instrukcí, program skočí o imm2 instrukcí dopředu.	42 10 00 00 - zmrzne program na 16 instrukcí. Pokud nikdo nezavolá teleport, skočí o 0 dopředu (tedy na tu samou instrukci a zavolá teleport znovu).
50 imm(16b) ??(8b)	BOMB	Každým spuštěním se hodnota imm sníží o 1 – dojde k přepsání instrukce v paměti. Pokud byla hodnota 0, nelze snížit a program bude ukončen.	50 00 00 CD - program zanikne hned po spuštění této instrukce.

Například, následující kód se zacyklí:

```
69 00 00 00  ; NOP
11 00 00 01  ; REVJUMP reg0, reg0, 0x0001
```

Odevzdávání

Odevzdávat budeš zdrojový kód svého programu ve formátu značeném výše. Jak sis už mohl všimnout, ; je komentář, vše až do konce řádku se ignoruje. Prázdné řádky a mezery se také ignorují. Na každém

jiném řádku je potřeba mít přesně 8 číslic, které dohromady dají (hexadecimálně) 32 bitů každé instrukce.

Odevzdávání této úlohy je složitější kvůli způsobu vyhodnocování. Jelikož je úloha komplexní, je zapotřebí dát účastníkovi podrobnou zpětnou vazbu. Podrobné informace o běhu tvého programu půjde zjistit na fiks.soptik.tech.

Na odevzdávací web se registruješ tím, že odešleš soubor, který bude mít na prvním řádku `email:<fiks email>` a na druhém `heslo:<moje heslo>`. Email musí být totožný jako ten, pod kterým máš zaregistrovaný svůj FIKS účet. **Nastav si jiné heslo, než používáš kdekoliv jinde!**

Po odevzdání je možné se svým emailem a zvoleným heslem přihlásit na web fiks.soptik.tech, kde si můžeš zobrazit výsledky svých odevzdání a zároveň stáhnout záznam toho, co se stalo. U „ostrých“ sfinga odevzdání budeš mít k dispozici pouze omezený log, který bude zobrazovat pouze tvoje akce – je to z toho důvodu, abys nemohl tak jednoduše kopírovat kód ostatních programů.

Web fiks.soptik.tech po přihlášení umožní i nahrát program přímo. Takovýto program bude spuštěn s jednoduchým protivníkem, který se stále jenom cyklí. Dostaneš kompletní výpis celého stavu prostředí, abys mohl debugovat svoje programy.

Pokud chceš debugovací prostředí ve kterém dělá protivník něco jiného, přepiš jeho kód sám :)

Pokud bys měl s odevzdáním nebo webem jakékoliv problémy, ozvi se na FIKS discordu nebo na `petr.stastny@fit.cvut.cz`.

Hodnocení

Body se udělují následovně (uvidíš je na hodnotícím webu):

- 0.01 bodu za úspěšné nastavení hesla
- celkem 1 bod za to, že odevzdáš validní program
- celkem 2 body za to, že porazíš protivníka, který se pouze cyklí.
- celkem 3-5 bodů za to, že porazíš protivníka, který na tebe bude aktivně útočit. Můžeš dostat i nižší počet bodů, pokud se mu zvládneš dost dlouho bránit, ale nepodaří se ti ho porazit.

Pokud dosáhneš alespoň 2 bodů, budeš zařazen do bitvy s ostatními účastníky.

Jednou týdně proběhne souboj mezi všemi účastníky, jejichž poslední odevzdaný program získal alespoň 2 body. Pokud se souboje budou účastnit alespoň 4 účastníci, dostaneš až 2 extra body v závislosti na svém výkonu. (Tyto body z různých týdnů se nesčítají.)

Po uzavření úlohy proběhne finální souboj, na základě kterého bude rozděleno až 5 bodů.

Maximum bodů, kolik můžeš získat, je 10.

Získané body uvidíš na portálu fiks.soptik.tech, do sfingy budou nahrané později po konci kola.

Příklad

Následující program spočítá faktoriál čísla 4 (vstup je uložen v registru 1, výstup bude v registru 2).

```
; reg1 = 4
; reg2 = 1
; while (reg1 != 0) {
;   reg2 *= reg1
;   reg1--
; }
; ; reg2 = 24
```

```
; Move 4 to register 1
07 10 00 04 ; reg1 = reg0 + 4 (reg 0 is always zero)
            ; This could also be achieved by using SETIMMLOW.
; Move 1 to register 2
07 20 00 01 ; reg2 = reg0 + 1
; Loop start
; if reg1 is zero, jump to end
10 10 00 04 ; if (reg1 == reg0) pc += 4 (skip the whole loop here)

  ; reg2 *= reg1
  03 21 00 00 ; reg2 *= reg1
  ; reg1--
  02 10 00 01 ; reg1 -= 1
  ; Jump to loop start
  11 00 00 03 ; If reg0 is equal to reg0, jump three instructions backwards
     ; (to loop start)
; LOOP END

69 00 00 00 ; NOP

; reg2 is now 24
```