A CPU EMULATOR FOR COURSE OF ASSEMBLY LANGUAGES

Lukáš Charvát
Master Degree Programme (2), FIT BUT
E-mail: xcharv03@stud.fit.vutbr.cz

Supervised by: Aleš Smrčka
E-mail: smrcka@fit.vutbr.cz

Abstract: The article discusses the design of an emulator of a CPU architecure instruction set aimed to assembly languages course. While most of nowadays emulators are architecture specific, the emulator proposed in the article aims at education and better understanding of assembly languages. The emulator easily allows user to define a purpose-specific architecture and an instruction set in order to perform operations upon it, and to display its current state.

Keywords: emulation, assembly language, instruction set

1 ÚVOD

Cílem projektu je vytvořit aplikaci schopnou názorně zobrazovat stav procesoru emulovaného na úrovni jazyka symbolických instrukcí spolu se schopností snadno definovat strukturu architektury procesoru a jeho instrukční sadu. Výsledek práce by měl sloužit k nasazení zejména ve výukovém, ale i experimentálním prostředí.

Při pohledu na některé ze zástupců současných emulátorů lze zjistit, že většina z nich je zaměřena zejména na maximální rychlost prováděné emulace. Toho bývá dosaženo zpravidla mapováním jednotlivých operačních kódů emulovaného procesoru na funkce implementačního jazyka, které operují nad dříve definovanou fyzickou reprezentací procesoru uloženou ve formě abstraktního datového typu, což typicky vede pouze k omezené možnosti měnit emulovanou architekturu. Obvykle tak emulátory poskytují možnost emulovat několik variant jednoho typu architektury (např. Aspectrum[1], PearPC[2]). Jedním ze specifických zástupců je emulátor MAME[3], zaměřující se na emulaci arkádových herních strojů, který dovoluje vytvářet uživateli emulovanou architekturu podle jeho potřeb na velice nízké úrovni abstrakce. Nabízí tak vysokou modularitu, která je ovšem vykoupena náročným popisem jednotlivých komponent emulovaného stroje.

2 POŽADAVKY NA SYSTÉM

Ze zaměření projektu vyplývají tyto základní požadavky na systém:

- jednoduchost – emulátor by měl být lehce pochopitelný bez nutnosti číst příručku nebo návod,
- schopnost emulovat co největší množství architektur – schopnost emulátoru emulovat současné architektury a adaptovat se na platformy nově příchozí,
- přívětivost pro začátečníky – emulátor by měl být schopen používat i uživatel bez hlubokých znalostí problematiky architektur počítačů nebo programování v asemblerech.

[1] `http://aspectrum.sourceforge.net`
[2] `http://pearpc.sourceforge.net`
[3] `http://mamedev.org`

3 NÁVRH EMULÁTORU

S ohledem na požadavky kladené na emulátor se nabízí rozdělení návrhu projektu na čtyři části. V první sekci definován jazyk pro popis emulované architektury. Podobně je učiněno v sekci druhé pro instrukční sadu a třetí pro grafický výstup. Poslední část návrhu se zaměřuje na celkový princip činnosti emulátoru.

Narozdíl od stávajících nástrojů nově vyvíjený emulátor předpokládá užívání dvěma typy uživatelů. Prvním typem jsou uživatelé definující architekturu a instrukční sadu emulovaného procesoru (vyučující). Typem druhým jsou uživatelé, kteří emulují programy a sledují vnitřní stav procesoru (studenti). Článek se dále zabývá pohledem na emulátor ze strany prvního typu uživatelů.

3.1 REPREZENTACE A POPIS ARCHITEKTURY

Pro dosažení schopnosti emulovat co nejvíce počítačových architektur musí být emulátor schopen reprezentovat základní prvky vyskytující se ve většině z nich. Z provedené analýzy vyplývá, že tyto prvky lze rozdělit z pohledu identifikace na tři typy:

- *jedinečné*, v architektuře jednoznačně identifikované názvem a bitovou šířkou (např. registr),
- *kolekce* (pole) prvků s přístupem pomocí indexu – identifikované názvem, bitovou šířkou a počtem prvků v kolekci (např. paměť),
- *symbolické*, které se v architektuře fyzicky nevyskytují, ale jsou součástí jiného prvku (např. část registru).

Emulátor musí být schopen provádět operace čtení a přiřazení (zápis) nejen nad prvkem architektury jako celkem, ale i nad jejich částmi tzv. *řezy*. Bitová šířka všech typů není omezena. Stejně tak některé virtuální architektury uvažují ve své specifikaci i nekonečný počet prvků kolekce (např. halda u Java Virtual Machine [1]). Veškeré prvky musí být emulátor schopen dynamicky vytvářet a rušit (opět s ohledem na architektury virtuální). Na základě výše uvedených požadavků na schopnosti jazyka byla navržena gramatika a interní reprezentace architektury postavená na objektově orientovaném návrhu. Jazyk generovaný vytvořenou gramatikou (dostupná na [3]) je podobně jako další navržené jazyky inspirovaný jazykem Python nabízející stručné a jednoduché vyjadřování. Na následujících řádcích je naznačena ukázka popisu architektury Intel x86 [2].

```
object eax = bits[32]         # Vytvoření 32bitového registru EAX
symbol ax = eax[0:15]         # Registr AX je částí registru EAX
array memory = bits[1024][8] # Vytvoření paměti 1024 x 8 bitů
eax[16:23] = memory[256]      # Načtení dat z paměti
```

3.2 REPREZENTACE A POPIS INSTRUKČNÍ SADY

Popis instrukční sady v sobě zahrnuje návrh prostředků reprezentujících zejména příkazy měnící stav emulované architektury. Mezi ně se řadí matematické a logické operace, procedury reprezentující sled těchto operací, podmíněné větvení v podobě podmínek a cyklů, deklarace pomocných proměnných pro výpočet a samotné popisy chovaní instrukcí představující mapování zápisu instrukce v jazyce symbolických instrukcí emulované architektury na operace nad dříve definovanými prvky této architektury. Pomocí výše uvedených prostředků je možné emulovat většinu instrukcí uživatelského režimu procesoru. Práce se nezabývá převodem strojového kódu do jazyka symbolických instrukcí (angl. *disassembling*), vazba na prvky řídící tok programu je realizována na úrovni grafického rozhraní. Na základě těchto požadavků byla opět vytvořena potřebná gramatika [3]. Na příkladu je uvedena ukázka popisu instrukce inkrementace zvoleného registru u architektury Intel x86.

```
instruction inc(REG32 r):                    # Definice instrukce
  tmp = r + 1                                # Inkrementace
  set_af(r, tmp); set_of(r, tmp)             # Příznaky AF, OF
  r = tmp[0:31]                              # Přiřazení
  set_pf(r); sf = r[31]; zf = (r == 0)       # Příznaky PF, SF, ZF
```

3.3 REPREZENTACE A POPIS GRAFICKÉHO VÝSTUPU

Data uchovaná v objektech reprezentujících prvky emulované architektury umožňuje emulátor zobrazit v rámci grafického výstupu aplikace. Vzhledem k výukovému zaměření projektu emulátor umožňuje prvky společného typu nebo podílející se na podobné funkci zobrazit společně. Emulátor také umožňuje zobrazit hodnoty prvků architektury v různých kódováních a formátech (např. binární, hexadecimální). Dále je nutno zachytit logickou vazbu mezi prvky emulované architektury umožňující vidět provázání jednotlivých elementů architektury z hlediska sémantiky (např. paměť a registr do ní ukazující). Logická vazba usnadňuje orientaci při práci s kódem emulovaného procesoru. Pro realizaci těchto požadavků byl vytvořen jazyk pro popis grafického výstupu. Následující příklad demonstruje logickou vazbu mezi pamětí a do ní odkazujícími hodnotami uloženými v registru ukazatele zásobníku a registru zásobníkového segmentu u architektury Intel x86.

```
pointer<memory> stack_top = (ss << 4) + sp  # Vrchol zásobníku
```

3.4 PRINCIP ČINNOSTI EMULÁTORU

Emulátor je řízen pomocí interpretu, který je navržen jako dynamický – změny se projevují po jednotlivých událostech (např. příkazech příslušného jazyka). Po definici prvků architektury a její instrukční sady umožňuje emulátor zápis kódu v jazyce symbolických instrukcí emulované architektury. Při vzniku uživatelem generované události v grafickém uživatelském rozhraní (např. příkaz zpracování další instrukce) se volá interpret instrukcí, který na základě překladových pravidel vzniklých při definici instrukční sady danou symbolickou instrukci rozpozná a vykoná příslušné operace nad prvky emulované architektury. Změna stavu prvků emulované architektury se promítá zpět do grafického uživatelského rozhraní, kde je zobrazena uživateli.

4 ZÁVĚR

Na základě výše uvedeného návrhu byly sestaveny příslušné gramatiky, s nimiž je možné generovat odpovídající jazyky [3]. Pomocí nich byla popsána architektura a instrukční sada procesoru Intel x86, která byla následně zprovozněna v prototypové verzi emulátoru implementovaného v jazyce Python. Grafický výstup bude zajištěn knihovnou Qt. Použití těchto nástrojů dovoluje provozovat emulátor na různých operačních systémech. Po dokončení implementační a testovací fáze by měl vzniknout nástroj schopný emulovat většinu dnešních a díky své univerzálnosti i budoucích architektur se zaměřením na výuku asemblerů.

REFERENCE

[1] Lindholm, T., Yellin F.: The Java Virtual Machine Specification, Second Edition, Prentice Hall PTR, 1999, ISBN 978-0201432947

[2] Intel 64 and IA-32 Architectures Software Developer's Manuals, Vol. 1 Basic Architecture, http://download.intel.com/design/processor/manuals/253665.pdf

[3] Charvát, L.: Emulace CPU pro výuku asemblerů, stránky projektu, http://www.stud.fit.vutbr.cz/~xcharv03/projects/aime