Posudek oponenta bakalářské práce

Student: David Hývl
Název: Simulátor MIPS procesoru

Znalost principu činnosti procesorů je důležitá ne jen pro ty, kdo nové procesory navrhují, ale i pro jejich uživatele, programátory. Umožňuje jim pochopit jakým způsobem je algoritmus realizovaný na úrovni hardware a jaká omezení pro propustnost z toho vyplývají a které operace a jejich kombinace jsou časově nákladné. Tato znalost pak umožňuje optimalizovat jak algoritmy tak slouží jako vodítko při výběru datových struktur. Pro seznámení s principy činnosti procesoru je pro studenty přínosné použít simulátor, který stav procesoru a vykonávání instrukcí zobrazuje. Vytvoření takovéhoto simulátoru s názorným zobrazením stavů a signálů bylo zadáním hodnocené bakalářské práce.

Vybraná procesorová architektura MIPS je dostatečně jednoduchá pro seznámení s problematikou a přitom umožňuje demonstrovat již základní techniky pro zvýšení propustnosti systému. Práce realizuje procesor s pětistupňovým zpracováním instrukcí včetně grafického zobrazení hodnot signálů na výstupu jednotlivých funkčních jednotek a registrů.

Práce začíná stručným popisem architektury MIPS. Stručnost je na místě, protože studijních textů s popisem existuje dostatek. Přesto si myslím, že před popisem jednotlivých formátů instrukcí R, I a J by bylo vhodné umístit tabulku nebo graficky znázornit umístění polí a zdůraznit společné pole pro operační znak a některé role polí registrů. Zpracování simulátrou je na této znalosti postavené a opis posunů a dalších operací nad instrukčním kódem v kapitole 3.4.7 by byl s odkazem na tuto tabulku názornější.

Student zvolil pro realizaci vhodnou grafickou knihovnu (Qt5), která zajišťuje programu velmi dobrou přenositelnost mezi různými operačními systémy a platformami. Oceňuji také snahu položit základy simulátoru na koncepci pojmenovaných objektů, které reprezentují signálové spoje a komponenty. Stejně tak pozitivně hodnotím snahu oddělit vlastní jádro simulující komponenty procesoru od grafického zobrazení. Zobrazení stavu procesoru a pamětí pak plně využívá plochy obrazovky. Zde bych ovšem důrazně vytkl použití rastrového obrázku pro zobrazení diagramu procesoru. Použití vektorového formátu (například Qt5 dokonale podporovaného SVG) by bylo vhodnější a šlo by i v některých případech s využitím parametrů použít přímo pro zobrazení hodnot a třeba i barvení spojů. Ještě výhodnější by bylo přímé navázání prvků modelu na systém jejich zobrazování a propojení. Pokud by bylo možné i zpětně s modelem interagovat, vznikl by již téměř obecný emulátor digitálních obvodů. Taková koncepce by ovšem byla nad rámec zadání a pro běžného studenta i nad rámec předpokládaného rozsahu práce.

Dále nepovažuji za uživatelsky přívětivé řešení režimu editoru zvlášť od režimu simulace. Použití záložek nebo okénka v rámci simulátoru by bylo vhodnější. Při překladu by pak bylo vhodné do rozšířených servisních údajů k jednotlivým instrukčním kódům uložit i odkazy na zdrojové řádky a při simulaci zvýrazňovat ne jen zpětný překlad, ale i řádku v okně, kde lze okamžitě modifikovat danou instrukci a v případě změny pak uživatele upozornit na potřebu nového překladu.

Řešení překladu instrukcí s využitím funkce strok() a postupného porovnávání řetězců v podmínkách if není vhodné. Funkce strtok využívá globálních proměnných přitom Qt5 a Stl nabízejí podobné funkce na bázi objektů a i pro případ omezení pouze na jazyk C lze doporučil buď použití strok_r() nebo v tomto konkrétním případě implementovat i vlastní řešení, než použít standardní funkci s nevhodným API. Popis rozpoznávaných instrukcí by pak měl být daný setříděným polem s binárním vyhledáváním nebo některou vyšší datovou strukturou (map, hash, tree) a to ne jen z důvodu rychlosti běhu, ale především z důvodu čitelnosti a rozšiřitelnosti. Zavedené řešení pro zpracovaní jazyků je pak z gramaticky a popisu generovaný tokenizer a parser (např. flex, lex, bison, yacc). Integrace překladu přímo do simulátoru sice zjednodušuje instalaci, ale volání externího assembleru a načítání binárního spustitelného formátu ELF by možná nebylo složitější na realizaci a s voláním z integrovaného editoru stejně pohodlné. Přitom knihovny libbfd

nebo readelf činí práci se souborem obsahujícím program jednoduchou a výhodou by byla i možnost nahrání a demonstrace kódu přeloženého kompilátorem jazyka C.

V implementaci simulátoru mi pak chybí funkce označení řádek kódu, kde se má kontinuální běh zastavit (break).

Kladně hodnotím, že program vyvíjený v prostředí MS Windows bylo možné s minimálními potížemi spustit na 64-bitové variantě systému GNU/Linux. Základní krokování sekvencí aritmetických instrukcí probíhalo v pořádku, ale instrukce podmíněného skoku byla ignorovaná, i když podmínka byla splněna. Reset procesoru vedl k hlášení o uvolnění již uvolněné paměti (double free) od GlibC, pravděpodobně mscrt.dll chyby ignoruje. Podle nástroje valkyrie+valgrind je se jedná o problém ve funkci reset() při volání funkce free_simple().

Do vlastního porovnání dostupných simulátorů je zařazeno přiměřené množství alternativ. Chybí ovšem přehled rozdílných koncepcí a možných nástrojů pro emulaci architektur na pro jiné než výukové účely. Jedná se výkonné simulátory určené pro běh celých operačních systémů vybavených někdy i s částečným překladem do nativního kódu za běhu (například QEMU, VirtualBox) a naopak pro prověření návrhu čipů a nebo systemizovaných jader použití technik simulace hardware na úrovni RTL nebo přímo jednotlivých hradel a trazistorů. Další zajímavou kategorií jsou emulátory herních konzolí. Například MAME-MESS, který mimo jiné systémy emuluje i několik systémů s architekturou MIPS. Pro emulaci periferních zařízení a logiky okolí procesoru používá i podobné řešení (knihovna NETLIST), jako je použité v práci a srovnání by bylo zajímavé a v době návrhu by bylo možné použít toto a podobná jiná řešení (např. na Qt založený simulátor elektronických systémů a obvodů Qucs) jako zdroje inspirace.

Práci **doporučuji k obhajobě**. Přesto, že považuji přístup studenta k problému za přiměřený znalostem získaným/poskytnutým mu v rámci bakalářského studia, tak pro množství připomínek, nedořešených problémů a výslednou použitelnost software mi nezbývá než práci hodnotit klasifikačním stupněm **dobře (C)**. Zároveň věřím, že práce a zkušenost přispěla k rozvoji schopností studenta a že další projekt již bude řešit s větším přehledem. Napovídá tomu i zhodnocení práce studentem v odevzdaném textu.

Otázka k případnému zamyšlení: Jakým způsobem by bylo možné rozšířit reprezentaci spojů WIRE a logiku funkce step_simple() tak, aby simulace nezávisela na pořadí uvedených výkonných funkcí bloků? Například, pokud by byl graf propojení editovatelný uživatelem nebo byl předepsaný v souboru.

V Praze, dne 28. 1. 2016

Ing. Pavel Píša, Ph.D.
Katedra řídicí techniky
Fakulta elektrotechnická
České vysoké učení technické