LANGUAGE C COMPILER BACK-END FOR SOFT-CORE MICROCONTROLLER

Jakub Horník

Master Degree Programme (2), FIT BUT

E-mail: xhorni00@stud.fit.vutbr.cz

Supervised by: Zbyněk Křivka

E-mail: krivka@fit.vutbr.cz

Abstract: A compiler allows us to describe an algorithm in a high-level programming language with a higher level of abstraction and readability than a low-level machine code. This paper describes design of a compiler back-end of subset of language C for 8-bit soft-core microcontroller Xilinx PicoBlaze-3.

Keywords: back-end, compiler, PCComp, PicoBlaze, Small Device C Compiler

1. ÚVOD

V současné době se při vytváření softwarového produktu, ať už ve formě komplexního programu, nebo jednoduchého skriptu, využívá výhradně některý z vyšších (vysokoúrovňových) programovacích jazyků. Avšak než je možné výsledný program spustit, je potřeba jej přeložit do formy, ve které může být vykonán na cílové platformě. Tento úkol má na starost program nazývaný *překladač* (či *kompilátor*). Zjednodušeně se tedy jedná o program, který přečte program v jednom jazyku (tzv. zdrojový jazyk) a přeloží jej do ekvivalentního programu v jiném jazyce (tzv. cílový program). Výstupním jazykem nejčastěji bývá nízkoúrovňový strojový kód cílové platformy. Překladač tedy umožňuje programátorovi popisovat algoritmus ve vysokoúrovňovém programovacím jazyce s vyšší mírou abstrakce a strukturovaností. Zároveň je zdrojový kód kratší a čitelnější. Tato práce se zaměřuje na tvorbu překladače pro malý softwarový procesor s využitím především ve vestavěných systémech implementovaných na čipu FPGA.

2. ARCHITEKTURA PŘEKLADAČE

Na nejvyšší úrovni abstrakce můžeme architekturu překladače rozdělit na dvě části:

- Přední část (front-end) – část závislá na vstupním jazyce
- Zadní část (back-end) – část závislá na cílové platformě procesoru

U této struktury neprobíhá překlad přímo, ale je rozdělen do dvou kroků. Nejprve je zdrojový text překládán do mezikódu, a až poté z mezikódu do cílového jazyka. Výhodou této architektury je, že se kompilátor rozdělí na dvě relativně nezávislé jazykové transformace, přičemž překlad ze zdrojového jazyka do mezikódu je společný pro více cílových platforem a až typem zadní části se určuje cílová platforma. Navíc záměnou přední části, při zachování struktury mezikódu, docílíme podpory pro jiný zdrojový jazyk [1].

3. XILINX PICOBLAZE

PicoBlaze je kompaktní, velmi jednoduchý a cenově nenáročný softwarový osmibitový RISC procesor, navržený firmou XILINX pro využití v FPGA čipech rodiny Spartan-3 a Virtex. Mezi základní rysy procesoru patří 8bitová aritmeticko-logická jednotka, 16 osmibitových registrů pro všeobecné použití, programová paměť pro 1024 instrukcí a pouhých 64 bajtů datové paměti [2].

4. NÁVRH PŘEKLADAČE

Cílem této práce je vytvořit zadní část překladače pro procesor *PicoBlaze-3* za využití již existující přední části překladače jazyka C.

4.1. EXISTUJÍCÍ PŘEKLADAČ PCCOMP

Pro procesor PicoBlaze již existuje jednoduchý překladač z jazyka C, nazvaný *PCComp* (*PicoBlaze C Compiler*). Avšak tento překladač se již nevyvíjí a jeho poslední verze je z roku 2005. Zároveň trpí zásadními neduhy, které jeho využitelnost značně degradují.

Generování probíhá na principu zásobníkového kódu, což je vzhledem k absenci zásobníku a velikosti datové paměti neefektivní a v některých případech je výsledný kód zcela nefunkční. Zároveň překladač neprovádí žádný druh optimalizace. Dalším omezujícím faktorem je podpora maximálně 16bitového datového typu a pouze jednorozměrných polí. Není podporována ukazatelová aritmetika. Z toho mimo jiné vyplývá, že parametrem funkce nemůže být ukazatel nebo pole.

4.2. VOLBA PŘEDNÍ ČÁSTI PŘEKLADAČE

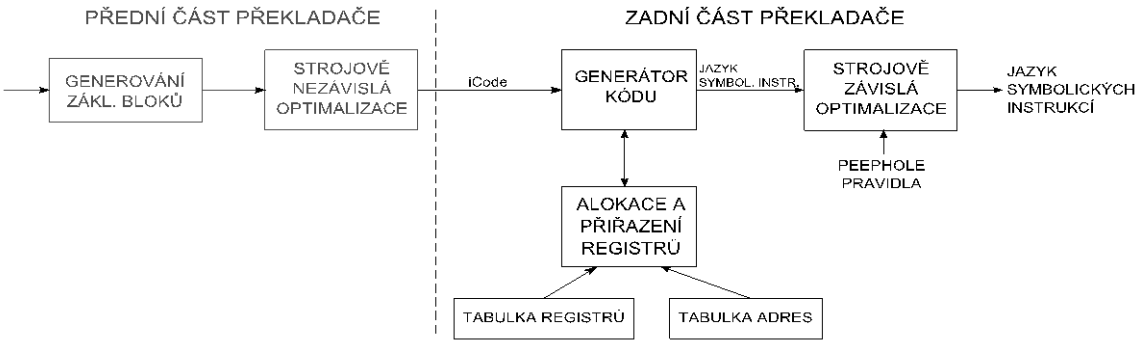
Jako nejvhodnější se jevila volba překladače *Small Device C Compiler* (SDCC). Tento překladač jazyka C byl vyvinut především pro 8bitové procesory a zároveň je distribuován pod licencí GNU General Public License. Překlad probíhá ve dvou krocích. V prvním kroku je generován mezikód jazyka, který je v SDCC označován jako *iCode*. Tato struktura obsahuje veškeré důležité údaje, které jsou potřeba pro generování cílového programu, zejména typ operace a seznam operandů. Ty tvoří další hlavní strukturu *operand*, kde jsou definovány příznaky (global, volatile apod.) a určen typ operandu jako symbolický operand (*symbol*), nebo literální hodnota (*value*). U symbolického operandu je důležitá zejména informace o životnosti proměnné, což umožňuje efektivně spravovat registry.

4.3. NÁVRH ARCHITEKTURY

Jak již bylo řečeno výše, překladač je založen na architektuře sestávající z přední části, závislé na vstupním jazyce, a zadní části, závislé na cílové platformě. Obrázek 1 znázorňuje návrh architektury zadní části našeho překladače (pravá část zobrazena sytě černou barvou) a její navázání na přední část překladače. Ta je pro zjednodušení zobrazena pouze jako generátor základních bloků spolu s modulem pro strojově nezávislou optimalizaci.

Prvním blokem zadní části je *Generátor kódu*, který na základě typu instrukce definované mezikódem provede vygenerování odpovídající posloupnosti instrukcí jazyka symbolických instrukcí.

Druhým blokem je *Alokace a přiřazení registrů*, jehož smyslem je udržovat informace o obsazenosti registrů a paměti dat konkrétními proměnnými a na základě těchto poznatků efektivně využívat registry v instrukcích generovaných prvním blokem. Případně při nedostatečném množství registrů se provede jejich uvolnění tak, aby to mělo co možná nejmenší dopad na výslednou efektivitu a výkonnost.



Obrázek 1: Architektura zadní části překladače.

Posledním blokem zadní části překladače je *Optimalizace cílového kódu*, který je založen na tzv. peephole optimalizaci popsané v [1]. Ten má za úkol na základě vhodně zvolených optimalizačních pravidel provádět zefektivnění cílového kódu.

Takto popsaný návrh, kdy bloky pro generování kódu a práci s registry spolupracují v rámci jednoho průchodu instrukcemi mezikódu, mohl být zvolen vzhledem k jednoduchosti samotného procesoru PicoBlaze a jeho instrukční sady. SDCC je ovšem navrženo pro komplexnější procesory jako např. PIC16, HC08 nebo MCS51 a samotné generování je zde rozděleno do dvou fází. Prvním průchodem se pouze přiřadí registry jednotlivým instrukcím mezikódu, a až při druhém průchodu se provede vygenerování výstupních instrukcí. Tyto průchody se provádí pro každou funkci jazyka C zvlášť v pořadí shodném s jejich uvedením ve zdrojovém kódu, nikoliv tedy pro zdrojový kód jako celek. To je ovšem při řešení námi navrhnutým způsobem nežádoucí, jelikož už od počátku postrádáme informace o všech globálních proměnných a znemožňuje to efektivně rozvrhnout datovou paměť, které má procesor PicoBlaze k dispozici velmi omezené množství. Řešením bylo při volání zadní části překladače pouze ukládat sekvence *iCode* instrukcí, patřících vždy k jedné samostatné funkci, do obousměrně vázaného seznamu a samotné generování spustit až po zpracování celého zdrojového programu. Bez nutného zásahu do implementace přední části ale informaci, že se jedná o poslední sekvenci *iCode*, nemáme nijak k dispozici. Samotné volání generátoru bylo proto přesunuto až do konečné fáze SDCC (nazvané *glue*), která spojuje všechny vygenerované bloky kódu jazyka symbolických instrukcí do výsledného souboru, čímž jsme se vyhnuli modifikaci přední části překladače.

Vzhledem k tomu, že procesor PicoBlaze neobsahuje žádné dedikované instrukce pro násobení, dělení a operaci modulo, bylo nutné tyto operace implementovat pomocí instrukcí sčítání a posunu v případě násobení, a instrukcí odečítání a posunu v případě operací dělení a modulo. Princip těchto algoritmů je popsán v [3]. Menší komplikací jsou operace se znaménkovými čísly, kdy je třeba před samotnou operací zkontrolovat, zda je operandem záporné číslo a případně jej převést na kladné, s tím, že se tato změna poznamená. Po provedení samotné operace se poté přihlédne ke znaménkům vstupních operandů a na základě znaménkových pravidel u násobení a dělení se patřičně upraví znaménko výsledku operace.

5. ZÁVĚR

Tento příspěvek popsal návrh architektury zadní části překladače podmnožiny jazyka C pro procesor PicoBlaze. Nový překladač má za cíl eliminovat nedostatky již existujícího překladače popsaného v sekci 4.1, z nichž většina již byla odstraněna, částečně i vhodnou volbou přední části překladače. Některé jsou však vzhledem k jednoduchosti instrukční sady procesoru PicoBlaze neimplementovatelné, či alespoň velmi obtížně. Zde můžeme například jmenovat plnou podporu ukazatelové aritmetiky nebo čísel s plovoucí desetinnou čárkou.

PODĚKOVÁNÍ

Tato práce byla podpořena projektem MŠMT 2C06008 „Virtuální laboratoř aplikace mikroprocesorové techniky“ a výzkumným záměrem MSM 21630528.

REFERENCE

[1] Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, Second Edition. Pearson Education, 2007. ISBN 03-215-4798-5.

[2] XILINX: PicoBlaze 8-bit Embedded Microcontroller User Guide v2.0, dostupné na http://www.xilinx.com/picoblaze [citováno 24. 2. 2011].

[3] Patterson, D.A., Hennessy, J.L.: Computer Organization and Design: The Hardware/Software Interface, Fourth Edition. Morgan Kaufmann, 2008. ISBN 01-237-4493-8.