Specifikace kompilátoru C pro PicoBlaze

Luboš Lorenc

22. března 2007

1 Výchozí informace

1.1 PCComp

PCComp je klon jednoduchého kompilátoru C pro procesor PicoBlaze. Zdrojové kódy nejsou dostupné a kvalitní aktuální dokumentace také ne. Jediným vodítkem pro určení vlastností PCCompu byl autorův manuál z června 2005 (aktuálně nejsou dostupné ani stránky projektu). Z něj byly určeny následující vlastnosti PCCompu:

- Generování jistého klonu zásobníkového kódu — velmi neefektivní v případě procesoru s množstvím registrů a absencí zásobníku.
- Zdrojový program má právě jeden soubor. Je však možné vkládat hlavičkové soubory, které mohou obsahovat i programový kód (poněkud matoucí a nestandardní).
- Jednoduchý preprocesor podporující pouze nejnutnější direktivy.
- Absence optimalizací kódu, generovaný kód je velmi neefektivní. Ukázka generovaného kódu pro součet dvou lokálních proměnných má cca 35 instrukcí. Stejný kód by přitom šel přeložit na méně než 10 instrukcí.
- Práce s proměnnými je v podstatě omezena pouze na použití globálních proměnných. Kód generovaný pro práci s lokálními proměnnými je natolik neefektivní, že i velmi jednoduchá funkce zkonzumuje veškerou programovou paměť.
- Jsou podporována jednorozměrná pole.
- Částečně je podpořeno použití ukazatelů, nikoliv však ukazatelová aritmetika.
- Je podporován 16ti bitový `int`, ale explicitní typové konverze jsou podpořeny jen částečně.
- Podmínky u podmíněných výrazů jsou vyhodnocovány nestandardním způsobem (s největší pravděpodobností se jedná o derivát pascalovských konvencí).
- Parametrem funkce nemůže být ukazatel, nelze předávat pole.

1.2 Picoblaze

PicoBlaze je 8-bitový softcore procesor vyvíjený firmou Xilinx. Jeho hlavním využitím je realizace různého stavového řízení pro FPGA v asembleru tohoto jednoduchého a malého procesoru.

1.2.1 Základní vlastnosti

- Paměť programu: 1k instrukcí pevné délky (18 bitů)
- 16 osmibitových registrů
- 64 bytů datové paměti
- Příznaky `ZERO` a `CARRY`
- Jeden vstup pro externí přerušení

Většina instrukcí používá jako operandy registry, případně registr a konstantu. Pro přístup k paměti slouží dedikované instrukce `FETCH` a `STORE`. Všechny výpočty probíhají osmibitově, ale jsou poskytovány instrukce (například `ADDCY` a `SUBCY`), které k výpočtu používají příznak `CARRY` a tím umožňují poměrně efektivně implementovat podporu pro šestnáctibitový datový typ (sečítání i odečítání pak trvá 2 strojové cykly). Ačkoliv je instrukční sada poměrně malá, jsou instrukce dobře voleny a lze poměrně efektivně implementovat i podporu pro násobení a dělení s dobou výpočtu cca do 60 strojových cyklů.

1.2.2 Uložení proměnných

Pro uložení proměnných je k dispozici 64 scratchpad bytů RAM. Její adresování může být buď přímé nebo nepřímé přes obsah registru. Tím je umožněno implementovat podporu lokálních proměnných funkcí a polí. Není však poskytován žádný speciální bázový registr. Bude tedy zřejmě nutné jeden registr vyhradit jako bázový a skutečnou adresu za běhu počítat přičítáním ofsetu k tomuto registru.

Vzhledem k malému rozsahu překládaných programů a předpokladu nepoužívání rekurzivních funkcí lze lokální proměnné deklarovat v paměťové třídě `static` a tím překladači umožnit výpočet adres již v době překladu, použít přímé adresování a eliminovat tak nutnost výpočtu adresy za běhu.

1.2.3 Přerušení

Přerušení je řešeno tak, že po přijetí požadavku PicoBlaze zamaskuje přerušení, uloží na zásobník obsah `PC` a do `PC` uloží hodnotu 0x3FF, kde musí být uložena instrukce skoku na skutečnou adresu rutiny obsluhy přerušení. Pro návrat z přerušení pak existuje speciální instrukce, která navíc umožňuje opětovně povolit přerušení.

Obsluhu přerušení bude vhodné řešit jako funkci (například `void interrupt(void)`). Pomocí direktivy `pragma` se překladači oznámí, že tato funkce provádí obsluhu přerušení. Pro povolení přerušení při návratu lze vytvořit jednoduchou interní nebo knihovní funkci (například `void enable_interrupt(void)`. Kompilátor volání této funkce přeloží na odpovídající instrukci návratu z přerušení po dosažení příkazu `return` nebo konce funkce `interrupt`.

Dále bude nutné poskytnout funkci umožňující maskovat přerušení kdekoliv v programu, například `void set_interrupt(int enable)`. V závislosti na hodnotě parametru kompilátor provede vygenerování odpovídající instrukce.

1.2.4 Zásobník

Zásobník návratových adres je schopen uložit 31 adres a poté se začne cyklicky přepisovat. PicoBlaze zároveň nepodporuje žádný způsob, jak tento zásobník z programu přímo modifikovat. Bude tedy možné používat rekurzivní funkce, ale bude jen na programátorovi, aby si ohlídal maximální hloubku zásobníku.

2 Základní specifikace

Jelikož PCComp překladač obsahuje značné množství chyb a problematických pasáží, tak pracujeme na kvalitnější implementaci překladače jazyka C pro PicoBlaze nazvaný PBCC (PicoBlaze C Compiler).

2.1 Datové typy

- Datové typy `char`, `short` a `int` budou mít 8 bitů.
- Datové typy `long int` a `long long int` budou mít 16 bitů.
- Datový typ `float` bude 32 bitů IEEE.
- Datové typy `double`, `long double` a `long long double` budou 64 bitů IEEE.
- Datové typy s pohyblivou řádovou čárkou budou plně analyzovány, ale nebude se pro ně zatím generovat kód.
- Podpora polí bude realizována podle standardu bez omezení počtu indexů.
- Podpora struktur a unionů bude provedena podle standardu.
- Výčtový typ bude podpořen minimálně pro 1023 konstant, ale uměle omezen na 256 konstant — bude reprezentováno jedním bytem.
- Ukazatele budou podporovány dle standardu včetně ukazatelové aritmetiky.

2.2 Volání a parametry funkcí

- Bude dodržen minimální limit 127 formálních parametrů v deklaraci funkce a 127 argumentů při volání funkce požadovaný standardem.
- Není k dispozici zásobník, takže předávání argumentů bude probíhat částečně v registrech, zbytek přes scratchpad RAM nebo externí logiku. Použití registrů bude do jisté míry limitováno, protože efektivita výpočtu zbývajících argumentů bude do jisté míry přímo úměrná počtu použitelných registrů. Na úkor počtu „rychlých“ statických lokálních proměnných tedy bude nejspíše nutné část scratchpad RAM zabrat pro vytvoření zásobníku, na kterém by poté mohly být alokovány i „automatické“ lokální proměnné.

- Podpora speciální funkce pro obsluhu přerušení bude řešena pomocí direktivy `pragma`.
- Funkce `main` bude deklarována pouze takto: `int main(void)`. Po dosažení příkazu `return` bude návratová hodnota uložena na specifikované místo (registr, port) a program skončí v nekonečné smyčce. Pro specifikaci místa uložení bude použita direktiva `pragma`.

2.3 Základní aritmetické operace

Až po úroveň generátoru kódu budou plně analyzovány a podporovány všechny operace podle standardu. V generátoru kódu budou prozatím implementována následující omezení:

- Plně podporované budou tyto operace (pokud není uvedeno jinak, lze realizovat i pro 16 bitů):
  - Sčítání a odčítání včetně post- a pre-inkrementace
  - Bitové operace `AND`, `OR` a `XOR`
  - Operace posunu
  - Doporučuji přidat podporu pro operace rotace na 8mi bitech, která v C standardně není (využití v kruhových registrech a podobně). Je potřeba vymyslet vhodné operátory. Návrh: `<<<` a `>>>`.
  - Logické operace `AND` a `OR`. Podpora pro 16 bitů bude trochu složitější než u ostatních operací, ale je realizovatelná.
- Částečně podporované budou tyto operace
  - Násobení a dělení — podpora bude řešena programově pomocí násobicí a dělicí rutiny. Každé použití některého z operátorů `*`, `/`, `%` bude přeloženo jako volání této rutiny. Dále doporučuji přidat vestavěnou funkci `void div(int op1, int op2, int *quotient, int *reminder)`, která bude použitelná v případě, že bude potřeba vypočítat jak podíl, tak zbytek. Obě hodnoty tak budou k dispozici po jediném výpočtu dělení.

2.4 Knihovní funkce

Podrobnější popis bude doplněn později.

Prozatím je známo, že bude nutné poskytnout funkce pro maskování přerušení a přístup k portům.

3 Obecný popis kompilátoru

V této sekci jsou podrobněji zpracovány pouze preprocesor a lexikální a syntaktická analýza, ostatní části se budou průběžně vyvíjet, zatím jsou uvedeny pouze předpoklady, jak je řešit.

3.1 Preprocesor

- Preprocesor bude implementován co nejvíce podle standardu.
- Preprocesor bude řešen jako samostatná jednotka. Výstupem bude textový soubor v pevném formátu. Každý řádek bude nejprve obsahovat jméno zdrojového souboru, poté číslo původního řádku a pak vlastní zdrojový text. Jména souborů budou do lokalizačních informací vypisována po převedení všech řídicích a dalších speciálních znaků na podtržítka. Lokalizační informace tak budou tvořit kompaktní celek, který bude možné snadno identifikovat jako speciální lexém při následné lexikální analýze.

3.2 Lexikální analýza

- Klasická lexikální analýza s využitím nástroje Flex.
- Bude implementováno vlastní udržování informací o pozici ve vstupním souboru na základě informací doplněných preprocesorem.

3.3 Syntaktická analýza

- Použití bisonu, generování syntaktického stromu.
- Prozatím nebude implementována podpora pro zotavení, stejně v případě bisonu nebývá příliš produktivní, později lze dodělat.
- Poznámky k syntaktickému stromu přímo nesouvisející se syntaktickou analýzou:
  - Na úrovni syntaktického stromu provést linkování
  - V první verzi asi ne, ale později bude nutné provést důkladnou optimalizaci syntaktického stromu, především vyhledávání a eliminaci podobných podstromů a přeuspořádání aritmetických operací.
  - Průchod syntaktickým stromem bude pravděpodobně řešen rekurzivně, implicitní zásobník lze využít i jako sémantický. Za cenu jistého zpomalení se dá ušetřit práce se sémantickým zásobníkem.

3.4 Tabulka symbolů

- Zásobníková struktura AVL stromů
- Libovolný počet úrovní zanoření
- Informace budou uchovány po celou dobu překladu
- Odpovídající úrovně tabulky symbolů budou provázány se syntaktickým stromem pro rychlejší přístup k informacím o symbolech při víceprůchodovém zpracování.

3.5 Generování kódu

- V první verzi „klasická“ tabulková metoda přidělování registrů v základních blocích.
- Generování do assembleru KPCSM3, později lze dodělat i podporu pro pBlazeIDE, případně vytvořit úplně samostatný konvertor — formáty se liší jen v několika drobnostech.