

Popis mikrokontroléru MC68332 a využití GNU vývojového prostředí systému Linux

Pavel Píša (pisa@cmp.felk.cvut.cz)

10. července 1999

1 Popis mikrokontroléru

Obvod MC68332¹ je 32 bitový vysoce integrovaný mikrokontrolér, který kombinuje výkonné jádro s rozsáhlým vstupně výstupním subsystémem. Jednotlivé moduly jsou propojeny vnitřní sběrnici. Základním modulem je 32 bitová CPU (CPU32 [2]) odvozená z řady procesorů 68000, jedná se o modifikovanou verzi 68020.

Další moduly jsou

- system integration module (SIM) modul vnitřního a vnějšího propojení
- time processor unit (TPU) výkonný časovací kontrolér
- queued serial module (QSM) sériová rozhraní
- 2-Kbyte TPU SRAM module (TPURAM) paměť použitelná pro hlavní procesor nebo pro uživatelský mikrokód pro TPU

Zdroj hodinového signálu může být buď vnější signál s úrovní TTL nebo interní násobička s fázovým závěsem s referenčním krystalem 32.768-kHz. Ve druhém případě je možné měnit frekvenci procesoru i za běhu pouze změnou požadovaného poměru vnitřního a referenčního hodinového signálu.

Spotřeba je vzhledem k použité technologii HCMOS velmi malá a statická architektura registrů a paměti umožňuje její snížení na minimum při zastavení systémových hodin instrukcí low-power stop (LPSTOP). Okamžitý

¹ Referenčním zdrojem informací je manuál firmy Motorola viz [1].

stav registrů a paměti zůstane zachován do příchodu některé z očekávaných událostí.

1.1 Přehled vlastností jednotlivých modulů

System Integration Module (SIM)

- Obstarává propojení vnitřní (mezimodulové) a externí sběrnice
- Obsahuje logiku programovatelných chipselectů
- Umožňuje ochranu systému
- Obsahuje kontrolní čítač watchdog, hlídání správné hodinové frekvence, monitor systémové sběrnice
- Systémové hodiny mohou být odvozeny od 32.768-kHz krystalu, výsledkem je pak nízká spotřeba
- Obsahuje testovací/ladící logiku pro výrobní a uživatelské testování a pro vývoj

Central Processing Unit (CPU)

- Strojový kód shora kompatibilní s procesory MC68000 a MC68010
- Nové instrukce pro řídicí aplikace
- 32-bitová architektura
- Umožňuje ve spolupráci s vnější MMU (Memory Management Unit) implementovat virtuální paměť
- Rychlé provádění cyklů obsahujících jednu instrukci
- Instrukce pro práci a interpolaci tabulek
- Vylepšené zpracování vyjímek pro řídicí aplikace
- Podporuje trasování do změny toku instrukcí (návrat, volání, ...)
- Obsahuje vstup pro vnější signál hardwarového breakpointu a kompletní logiku pro ladění Background Debug Mode
- Plně statická činnost umožňuje snižování a i zastavení hodin procesoru

Time Processor Unit (TPU)

- Obsahuje vlastní řadič mikrokódu pracující nezávisle na CPU32
- 16 nezávislých, programovatelných kanálů a pinů
- Každý kanál může vykonávat libovolnou časovou funkci
- Více kanálů může být vzájemně synchronizováno nebo může vytvářet složitější funkci využívající více pinů
- Dva čítače času s programovatelnými předděličkami
- Volitelnou prioritu jednotlivých kanálů

Queued Serial Module (QSM)

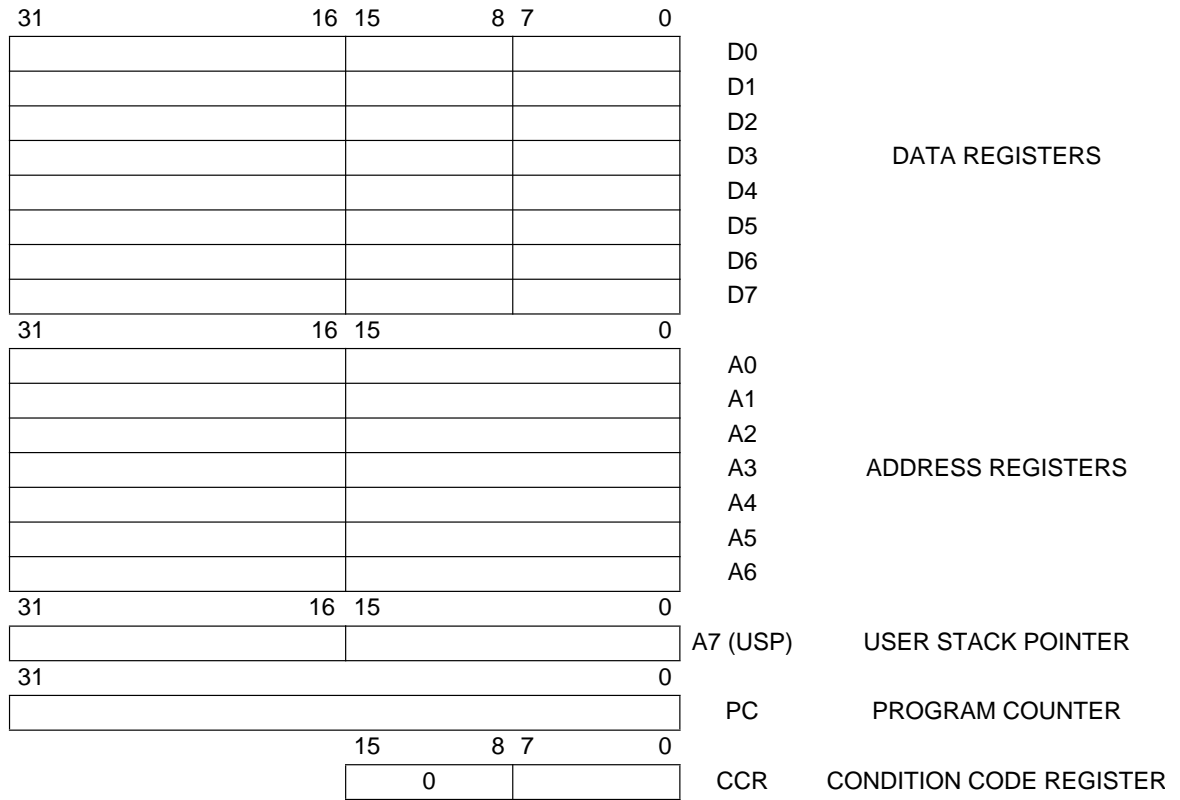
- Obsahuje sériový interface (SCI), univerzální asynchronní vysílač a přijímač (UART) s volitelným módem, rychlostí a paritou
- Sériový interface pro připojení periférií s 80-Byte RAM, který umí provádět až 16 přenosů automaticky
- Vstupy/výstupy s dvojí funkcí
- Cyklický mód, 8 až 16 bitů na jeden přenos

Static RAM Module s možností využití pro emulaci TPU (TPURAM)

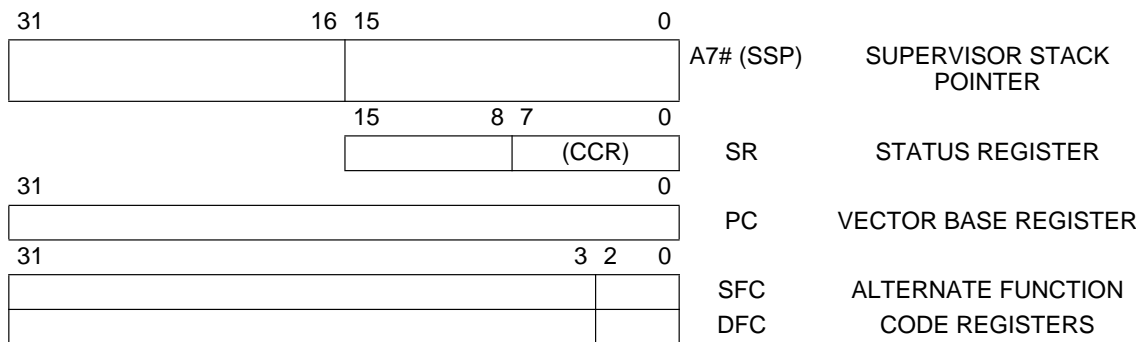
- 2-Kbyte statické paměti RAM
- Může být využita jako normální rychlá RAM nebo pro emulaci mikrokódu TPU

1.2 Stručný popis CPU32

Procesor je založen na architektuře CISC (Complex Instruction Set Computer). Jádro procesoru je plně 32 bitové, vnější datová sběrnice je pouze 16 bitová. Navenek je přístupných také pouze 24 adresových linek (možnost adresovat 16 MB paměti). Adresový prostor je lineární, bez jakýchkoli omezení a segmentací, pro periférie není vyhrazen speciální prostor. Systémová paměť (včetně boot ROM) i periférie mohou být organizovány 8 i 16 bitově. Pro 16 i 32 bitové přístupy do paměti platí, že adresa musí být sudá. Veškeré instrukce jsou 16 bitové s možností rozšíření o další 16 bitová slova. Instrukce a zásobník musí být také zarovnané na sudé adresy.



Obrázek 2: Uživatelský model CPU32



Obrázek 3: Systémový model CPU32

1.2.1 Registry a režimy procesoru

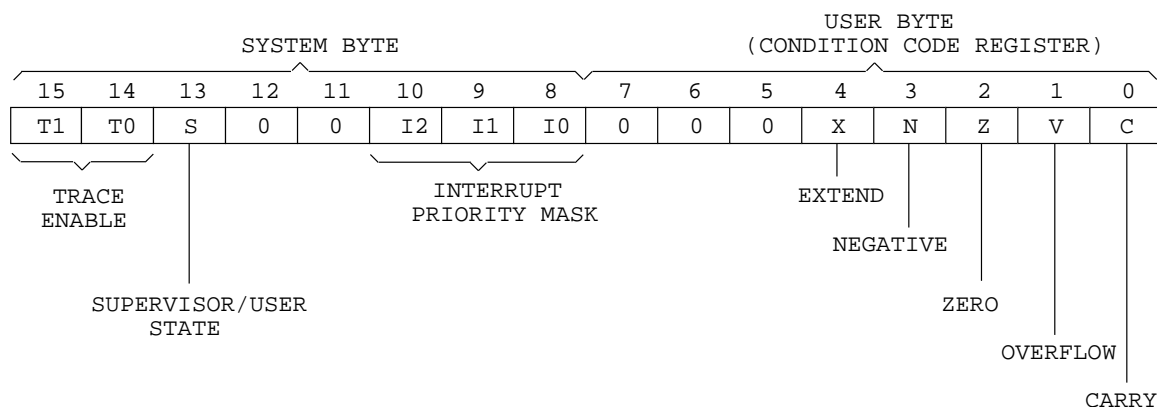
16 třicetidvou bitových registrů je rozděleno na 8 datových (D0-D7) a 8 adresových (A0-A7). Datové registry lze použít pro operace s 32, 16 a 8 bity. Adresové registry umožňují pouze 32 bitové a 16 bitové operace a jsou-li použity jako cíl 16 bitové operace je hodnota automaticky znaménkově rozšířena na 32 bitů. Datové registry lze použít k přístupu pouze v kombinaci s adresovým registrem jako index. Adresové registry lze použít jako index i jako zdroj báze adresy.

Jediný z těchto registrů, který má speciální určení, je registr A7, který slouží jako ukazatel zásobníku. Tento registr je zdvojen, A7 (USP) ukazuje do zásobníku uživatelského programu, A7' (SSP) je použit v systémovém režimu (supervisor). Jediný možný přechod z uživatelského do systémového režimu je pomocí vyjímek (softwarově vyvolané k tomu určenou instrukcí, vnější přerušení a způsobené chybami při provádění instrukcí - ochrana paměti, ilegální instrukce atd.). Každé vyjímce je přiřazen jeden z 256 třicetidvou-bitových vektorů uložených v paměti od adresy uložené v registru VBR (po startu procesoru obsahuje 0). Vnější přerušení mohou mít prioritu 0 až 7 (priorita 0 složí k zakázání přerušení) a mohou v potvrzovacím cyklu předat hodnotu vektoru přerušení.

Stavový registr procesoru CPU32 je rozdělen na dvě části, systémovou část a podmínkový registr (CCR), který je přístupný i z uživatelského režimu. Nevyužité bity stavového registru jsou čteny jako 0 a měly by také být také tak zapisovány pro zachování kompatibility s novějšími procesory. Systémová část stavového registru obsahuje dva bity pro řízení trasování (T1 a T0), příznak systémového režimu (S) a masku priority přerušení (IP2 až IP0). Povolení trasování způsobí po naplnění SR ze zásobníku a návratu instrukcí RTE do trasovaného programu vyvolání vyjímky po provedení jedné instrukce. CPU32 umožňuje i trasování, při kterém dojde k vyjímce pouze při změně toku vykonávaných instrukcí (například skok, návrat, volání). Masky přerušení povoluje procesoru přijmout pouze vnější požadavky na přerušení s vyšší prioritou. Vyjímku představuje pouze vnější úroveň 7, která je použita pro nemaskovatelná přerušení.

Bity podmínkové části (CCR) stavového registru jsou plněny logickými, aritmetickými a bitovými operacemi a operacemi rotací a posunů. Příznak X slouží k rozšiřování operací na operandy větší délky než 32 bitů. Příznaky N a Z informuje o záporném/nulovém výsledku předchozí operace. Indikace přetečení a přenosu jsou uloženy do příznaků V a C.

Většina kompilátorů využívá registr A6 jako ukazatel na rámec zásobníku aktuální funkce. Registr A5 bývá použit pro kód nezávislý na svém umístění - PIC (Position Independent Code). Registr D0 a pro 64 bitové výsledky



Obrázek 4: Stavový registr

i D1 většinou slouží pro předání návratové hodnoty funkcí.

1.2.2 Adresace - konvence

Pro přístup k operandům slouží 14 režimů adresace. V literatuře je užívána následující konvence pro popis režimů adresace a instrukcí.

EA efektivní adresa

An adresový registr n, například **A3**

Dn datový registr n, například **D3**

Rn libovolný z datových a adresových registrů

Xn.SIZE*SCALE index registr, libovolný datový nebo adresový registr

SIZE velikost indexu **W** (16 bitový) nebo **L** (32 bitový)

SCALE měřítko - násobitel indexu 1, 2, 4 nebo 8

PC čítač programu

SR stavový registr

SP ukazatel zásobníku (**A7 - USR** nebo **SSR**)

CCR podmínkový registr, nižší byte **SR**

USP ukazatel zásobníku v uživatelském režimu

SSP ukazatel zásobníku v systémovém režimu

dn ofset, délky n bitů

bd báze adresy až 32 bitů

L délka 32 bitů (long-word)

W délka 16 bitů (word)

B délka 8 bitů (byte)

(An) závorky určují adresaci obsaženou hodnotou

1.2.3 Režimy adresace

Následuje krátký výčet jednotlivých režimů adresace CPU32

Rn obsah datového nebo adresového registru

(An) obsah paměti na adrese **An**

(An)+ obsah paměti na adrese **An** s následnou inkrementací registru o hodnotu danou délkou operandu

-(An) nejdříve dojde k dekrementaci registru o délku operandu a pak je registr použit k adresaci

(d16,An) adresový registr s 16 bitovým znaménkovým posunutím

(d8,An,Xn) adresový registr s 8 bitovým znaménkovým posunutím a přičtením indexového registru

(bd,An,Xn*SCALE) adresa je vytvořena ze součtu adresového registru s indexovým registrem násobeným měřítkem **SCALE** a bázovým posunutím délky až 32 bitů

(xxx).W 16 bitová absolutní adresa

(xxx).L 32 bitová absolutní adresa

(d16,PC) adresace relativní k **PC** s šestnáctibitovým znaménkovým posunutím

(d8,PC,Xn) adresa relativní k **PC** s osmibitovým znaménkovým posunutím a přičteným indexem

(bd,PC,Xn*SCALE) adresa relativní k **PC** s posunutím až 32 bitů a s indexem násobeným měřítkem

Pro úplnost jsou dále uvedeny i režimy adresace, které nejsou implementovány v jádře CPU32. Tyto režimy jsou implementovány pouze v procesorech 68020 až 68060. Znalost těchto chybějících režimů může být výhodná při hledání problémů s programy původně určenými pro výkonnější členy rodiny 680x0.

([bd,An],Xn,Od) adresu tvoří hodnota v paměti na adrese **An+bd**, ke které je přičteno posunutí a index

([bd,PC],Xn,od) totéž ale relativně k **PC**

([bd,An,Xn],Od) adresu tvoří hodnota v paměti na adrese **An+Xn+bd**, ke které je přičteno posunutí

([bd,PC,Xn],od) totéž ale relativně k **PC**

1.2.4 Instrukční soubor

Procesor CPU32 rozeznává asi 60 instrukcí, které po doplnění režimy adresace vytvářejí více než 1000 užitečných kombinací. Základní datové typy jsou bity, byty, čísla BCD, 16 bitová slova a 32 bitová dlouhá slova. Instrukční soubor obsahuje obvyklé aritmetické, logické, bitové, přesunové a řídicí instrukce. Ze zajímavých instrukcí procesoru CPU32 je možno považovat instrukce pro interpolaci hodnot v indexovaných tabulkách a rozšíření instrukcí pro násobení a dělení oproti procesoru 68010 o plně 32 bitové operandy a dokonce o možnost násobení dvou 32 bitových čísel s 64 bitovým výsledkem a dělení 64 bitového čísla číslem 32 bitovým.

Podrobnější informace o instrukcích a architektuře CPU32 lze nalézt v [2].

1.2.5 Výkonnější následníci CPU32

Při výběru určité rodiny procesorů je nutné uvažovat i o jejich předpokladech budoucího vývoje nebo náhrady za výkonnější členy a zachování alespoň částečné kompatibility s již použitými členy. Rodina procesorů 680x0 byla v minulosti založena velmi velkoryse, v instrukčním souboru bylo vyhrazeno místo pro nové instrukce a adresní módy, procesor od začátku pracoval s

32 bitovou lineární adresou atd. V současné době však byla výkonnostně architektura 680x0 překonána, proto se ve výkonných pracovních stanicích, personálních počítačích více prosazují jiné architektury (od firmy Motorola především PowerPC [5]). Avšak jednoduchost, elegance, snadná integrovatelnost více bloků na jeden čip a relativně nízká spotřeba předurčuje architekturu 680x0 k řídicím aplikacím. Pro aplikace, kde již výkon CPU32 obsaženého v 68332 nepostačuje, lze uvažovat o přidání některého z výkonnějších procesorů (68040, 68060), pro který může být 68332 vstupně/výstupním koprocetorem a nebo pouze vysoce integrovaným blokem periférií. Další alternativou od firmy Motorola jsou mikrokontroléry založené na procesoru PowerPC.

Vhodnost koncepce rodiny 680x0 pro řídicí aplikace vedla výrobce k úvahám o jejích dalších výkonnějších nástupcích. Výsledkem jsou mikrokontroléry založené na procesorovém jádře ColdFire 52xx [4], jehož architektura je založena přepracované a odlehčené verzi (pouze jedna pipeline) procesoru 68060. Procesor má přepracovaný instrukční soubor, který přímo odpovídá RISC jádru procesoru. Výsledkem je procesor s výkonem (až **50 MIPS @ 54 MHz**) srovnatelným s 68040 nebo Intel 486.

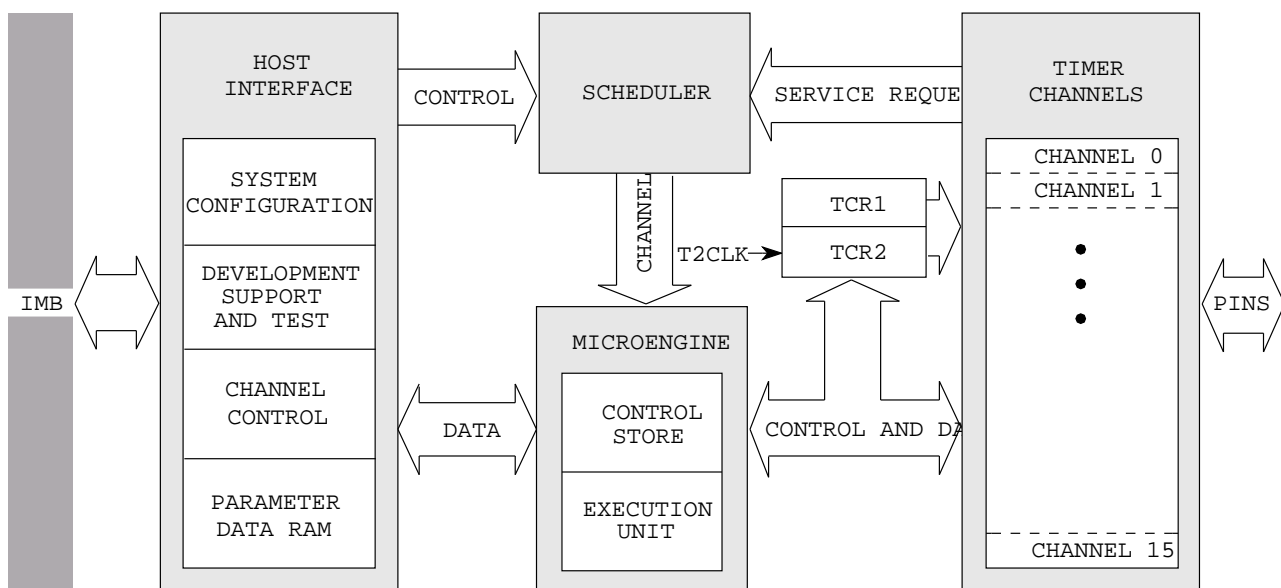
Všechny výše uvažované varianty a rodiny procesorů jsou podporovány kompilátorem GCC, proto přechody mezi jednotlivými procesory na úrovni rutin v jazyce "C" nejsou příliš cenově a časově náročné. Větším problémem jsou rozdíly v integrovaných perifériích (kromě kombinace 68060 a 68332 kde periférie zůstávají) a v návrhu nového hardware.

1.3 Integrační modul (SIM)

Konfigurační a ochranný blok řídí operační mód mikrokontroléru. Dále blok obsahuje ochranu proti zablokování nebo selhání sběrnice a ochranu proti zablokování software (watchdog). Zdroj systémových hodin generuje hodinový signál pro SIM, ostatní moduly připojené k vnitřní mezimodulové sběrnici (IMB) a pro vnější zařízení. Dále poskytuje generátor periodického přerušení pro řídicí a systémové funkce.

Interface vnější sběrnice vykonává přenosy mezi vnitřní sběrnici (IMB) a vnějším adresním prostorem. Blok chipselectů může dekódovat z požadované adresy až 11 libovolně použitelných výběrových signálů a jeden signál pro zaváděcí kód a případně i firmwarovou paměť (boot ROM). Pro každý chipselect je možno zvolit velikost dekódované oblasti (použita jako maska adresy), počáteční adresu, druhy přístupu (čtení, zápis, potvrzení přerušení) a datovou šířku připojené paměti nebo periférie (byte, word, liché byte, sudé byte).

Testovací blok obsahuje logiku pro testování mikrokontroléru při výrobě.



Obrázek 5: Časovací koprocessor TPU

O konfiguraci některých parametrů SIM je potřeba rozhodnout dříve, než dojde k načtení počáteční hodnoty PC a SSP z adresy 0 boot ROM. Tyto parametry jsou řízeny hodnotami přečtenými z datové sběrnice v době aktivního signálu RESET.

1.4 Časovací koprocessor (TPU)

Časovací koprocessor (TPU) umožňuje řídit a zaznamenávat i relativně komplikované časové děje. TPU obsahuje vlastní výkonnou jednotku, plánovač se třemi úrovněmi priorit, dvoubránovou paměť RAM pro zadávání konfigurace, příkazů a výměnu hodnot s CPU32, dvě časové základny a paměť ROM se standardními časovými funkcemi a 16 nezávislých kanálů. Funkce mohou být uživatelem přeprogramovány za cenu využití interních 2KB paměti RAM pro uložení mikrokódu TPU. Každý kanál je propojen s jedním vstupně výstupním pinem pro který je nezávisle definována časová funkce. Více kanálů může být vzájemně propojeno a tvořit komplikovanější funkce (například fázový dekodér a čítač pro inkrementální čidla polohy, referenční a měřenou frekvenci nebo sériové rozhraní). Každý kanál je doplněn hardwarem, který dovoluje současné vstupní i výstupní události na všech kanálech

1.5 Komunikační modul (QSPI)

Tento modul obsahuje podporu pro sériovou plně duplexní synchronní třídrátovou sběrnici (vodiče data in, data out a hodiny přenosu). Čtyři piny pro výběrové signály umožňují připojit až 16 periferních zařízení. Vlastní RAM umožňuje autonomní provedení až 16 přenosů s délkou 8 až 16 bitů, nebo přenos bloku až 256 bitů. Další možností je periodický cyklický mód činnosti, který je vhodný pro automatické obnovování čtených hodnot (například pro rozšíření systému o AD převodníky).

Blok pro sériovou komunikaci (SCI) je určen pro obvyklou asynchronní sériovou komunikaci. Přenášená slova mohou být délky 8 nebo 9 bitů a mohou být doplněna sudou nebo lichou paritou. Komunikace může být jak plně duplexní tak i poloduplexní s přenosovou rychlostí 64 až 256 kbaud pro frekvenci systémového hodinového signálu 16.78 MHz nebo 110 až 655 kbaud pro frekvenci 20.97 MHz. Přijímaný signál je filtrován od rušivých impulsů. Přijímač a vysílač jsou samostatně povolovatelné. Oba jsou vybaveny dvojitým datovým registrem. Přijímač umožňuje generování přerušení při příjmu dat s nastaveným devátým bitem, což je vhodné pro multiprocesorovou komunikaci.

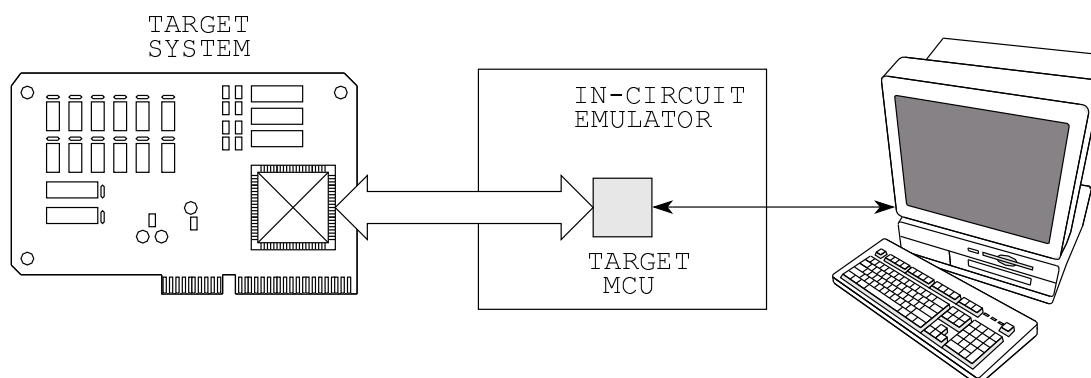
2 Přehled vývojových prostředků

Vývojové prostředky lze rozdělit do tří skupin. Hardwarové prostředky nutné pro vývoj a oživení vlastního procesorového systému. Prostředky pro vývoj aplikačního programového vybavení. Systém umožňující ladění takto vzniklých programů.

2.1 Hardwarové prostředky

Při ověřování návrhu hardware je často nutné testovat propojení a funkci základních funkčních bloků nově navrhovaného systému. Hardwarová emulace nebo alespoň záznam dějů v reálném čase jsou také nutné při ladění časově podmíněné obsluhy některých periférií a při analýze odezvy systému na vnější události.

Klasický obvodový emulátor je externí zařízení, které se připojí místo procesoru v cílovém zařízení. Emulátor většinou obsahuje oddělovací logiku, řadič pomocných sekvencí (obvykle s pomocným procesorem), interface pro komunikaci s nadřazeným počítačem a náhradu cílového procesoru. Náhradou může být buď původní procesor nebo speciální verze s vyvedenými vnitřními signály a nebo náhrada programovatelnými obvody se shodnou funkcí. Protože se jedná o systémy vyráběné v malém počtu je jejich cena vysoká.



Obrázek 6: Externí emulátor

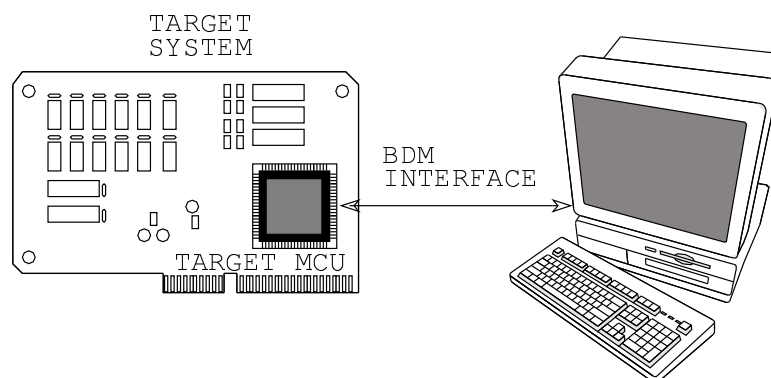
Protože je integrace dnešních procesorových obvodů již natolik vysoká a interní mikrořadič dostatečně výkonný, je možné přidat do mikroprogramů funkce umožňující při vývoji komunikovat přímo s jádrem mikroprocesoru. Takové řešení sice mírně zvyšuje složitost vyráběného obvodu, ale není pak nutné vyvíjet speciální emulační verze ani relativně složitou logiku externího emulátoru. V případě mikrokontroléru MC68332 je možné vnějšími signály zastavit výkon instrukcí programu a začít komunikovat s mikrořadičem přímo z nadřazeného počítače. Tento režim je nazýván BDM (Background Debug Mode). Mikrořadič dostává jednoduché příkazy třídrátovým sériovým interfacem a informuje o výsledcích těchto příkazů stejnou cestou.

Na následujících obrázcích zobrazen klasický přístup s externím obvodovým emulátorem (Obrázek 6) a systém využívající interního emulátoru procesoru MC68332 (Obrázek 7). Výhodou BDM je přítomnost hardwarové emulace v každé aplikaci a i na deskách, kde není mikrokontrolér umístěn v patici (např. SMD verze mikrokontroléru). Jistou nevýhodou proti některým externím emulátorům je nemožnost sledovat výkon instrukcí v reálném čase (rychlost krokování a emulace je dána především rychlostí komunikace přes BDM interface). Protože je BDM sériová komunikace synchronizována hodinovým signálem ovládaným z nadřazeného počítače, může být v mnoha případech rychlejší, než komunikace s klasickým externím emulátorem přes interface RS232.

2.2 Prostředky pro vývoj software

Úkolem těchto nástrojů je převod popisu požadované funkce do strojového kódu, který je schopen cílový procesor vykonávat.

Ve většině případů se jedná o překladače z vyšší úrovně (programovacího jazyka) popisu do úrovně nižší (postupně jazyk symbolických adres



Obrázek 7: Interní BDM emulátor

- assembler a strojový kód). Dnes již existují i prostředky pro převod popisu z blokových diagramů, překladače z matematických a simulačních jazyků (např. ze skriptů MatLabu nebo z schémat nakreslených v Simulinku). Pro lepší přenositelnost mezi různými cílovými architekturami je však většina nástrojů stavěna hierarchicky. Postupný překlad i u nejmodernějších návrhových systémů většinou využívá již zavedené standardní jazyky používané na většině platform. Těmi je jazyk “C” popřípadě “C++”. Pro každou cílovou platformu je tedy nutné vytvořit překladače z jazyka “C” do assembleru a z assembleru do cílového strojového kódu. Nejdůležitějším článkem řetězce je kvalita překladače jazyka “C”, jehož schopnost optimalizace rozhoduje o délce a rychlosti výsledného kódu. Většinou jedinou jinou cestou jak zrychlit cílový kód je psaní některých částí přímo v assembleru.

Jinou alternativou je využití cílového procesoru jako interpretru jiného popisu požadované funkce. Toto řešení však vyžaduje výše zmíněné prostředky alespoň k vytvoření interpretru. Výsledná rychlost systému je negativně ovlivněna spotřebováním části výkonu procesoru na analýzu popisu funkce.

2.3 Ladění aplikací

Posledními, ne však zanedbatelnými pomocníky při vývoji mikroprocesorového systému nebo při tvorbě aplikací, jsou prostředky pro kontrolu, hledání zdrojů chyb a rychlostního profilování aplikací. Pro počítače s obecným operačním systémem a uživatelským rozhraním (terminálem) se většinou využívá přímo vlastního systému a pomocných programů na cílové platformě.

U systémů určených k řídicím aplikacím většinou tento postup není možný. Systém má omezenou paměť, není vybaven terminálem, neumí zaručit současný běh laděné aplikace a kontrolního programu, není odolný proti chybám v laděné aplikaci. Podobné problémy je nutné řešit i při vývoji vlastního

operačního systému i u větších systémů.

V těchto případech je možné použít jedné ze dvou metod. První je využití již zmíněných hardwarových vývojových prostředků a nadřazeného počítače, jehož software umožňuje interpretovat stav procesoru na vyšší úrovni abstrakce. Tou je například možnost ladění na úrovni vyššího programovacího jazyka (tzv. source level debugging - ladění na úrovni zdrojového kódu). K tomu je nutné mít k dispozici zdrojové texty programů, informace o korespondenci řádek zdrojového textu s výsledným strojovým kódem, informace o okamžitém umístění lokálních a globálních proměnných v paměti a o reprezentaci různých datových typů v paměti a registrech cílového počítače.

3 GNU Tool Chain

Soubor programů pro vývoj aplikací, který vznikl v rámci projektu GNU. Obsahuje kompilátory jazyka “C”, “C++”, Objective-C, Pascal, Fortran, assembler a další vývojové prostředky, debuggery, profilery, knihovny funkcí, RT i obvyklé operační systémy.

3.1 Projekt GNU a FSF Inc

GNU projekt se zabývá vývojem alternativních volně šiřitelných programů především pro systémy UNIXového typu.

Hodně lidí asi slyšelo o **GCC**, **GDB**, **GNU Emacs** atd. Málo kdo však asi zná historii tohoto obrovského díla velkého množství nezávislých programátorů z celého světa, které je schopné co se kvality týče konkurovat i velkým komerčním firmám.

Vše začalo dopisem odeslaným 27. září 1983 ve 12:35:59 Richardem Stallmanem z MIT AI Lab v Cambridgi. Tento autor editoru Emacs a mnoha dalších interpretrů, kompilátorů, grafických knihoven atd. v dopisu píše, že se rozhodl vytvořit alternativu komerčních implementací **UNIXu**. Jeho základní pravidlo vyžaduje: “Používám-li nějaký program, který se mi líbí, musím mít právo ho vylepšit a poskytnout i ostatním, kterým se líbí“.

V současné době se o správu projektu **GNU** stará nezisková organizace pro software v obecném zájmu Free Software Foundation, Inc s E-mail adresou ‘*gnu@prep.ai.mit.edu*’.

Přesto, že se vlastní systém **GNU** příliš nerozšířil, umožnila existence volně šiřitelného programového vybavení vznik operačních systémů jako je FreeBSD, Hurd a Linux. Veškeré programy jsou napsány s maximální snahou o přenositelnost ve zdrojové formě na většinu UNIXových systémů včetně IBM OS/2 (projekt EMX). Velké množství je možné provozovat i v relativně

nekompatibilních prostředích jako je DOS (projekt DJGPP) nebo Win 95 a Win NT (projekt CygWin32).

V dalším výkladu bude postupně popsán způsob vystavění vývojového řetězce. Protože vyšší vrstvy jsou závislé na nižších vrstvách, je nutné začít řetězec budovat od překladače assembleru, linkeru a dalších utilit pracujících na úrovni relokovatelného a absolutního strojového kódu (souhrnně se nazývají BINUTILS - Binary Utilities).

Poté je možno přistoupit k přípravě překladače GCC jazyka "C", kompilaci knihoven a vlastního operačního systému a aplikací. Nakonec bude popsána příprava ladícího programu - debuggeru GDB.

3.2 Programy nutné pro přípravu vývojového prostředí

Následující nainstalované programy jsou nutné k přípravě GNU vývojového prostředí kompilací ze zdrojových textů:

make verze GNU programu make, který podle času modifikace jednotlivých souborů a databáze možných způsobů jejich překladu rozhoduje o tom, které soubory a jak je nutno přeložit a poté spojit do nových verzí výsledné aplikace

gcc GNU nebo i jiná verze překladače jazyka "C"

as překladač assembleru (nejlépe též verze GNU)

ld linker, program pro spojování částí strojového kódu do výsledného programu

ar knihovní program pro správu fragmentů strojového kódu

nm program pro výpis symbolických jmen použitých ve fragmentu strojového kódu

bison nebo **lex** lexikální analyzátor pro překlad lexikálního popisu do jazyka "C", jeho přítomnost není absolutně nutná, protože GCC je již dodáváno jak se zdrojovými texty pro bison tak s přeloženými soubory v jazyce "C".

m4 makro expander, jeho přítomnost není absolutně nutná

makeinfo systém pro přípravu dokumentace do hypertextového formátu info - jeho přítomnost je také volitelná

Veškeré zde zmiňované programy jsou standardní součástí všech distribucí systému **Linux**. Například v distribuci Slackware se jedná o soubor disket s názvem `development`. Většina programů je dostupná i v binární formě pro většinu platforem včetně **DOSu** (**DJGPP**).

3.3 BINUTILS assembler, linker a další utility

Balík obsahuje prostředky pro tvorbu a správu fragmentů strojového kódu pro cílovou platformu. Vstupními soubory jsou textové soubory obsahující v textové formě popis programu v jednotlivých instrukcích cílového procesoru. Vzájemné odkazy mezi funkcemi a daty jsou vyjádřeny pomocí symbolických jmen. Tento tvar bude dále zkráceně nazýván jménem překladače *assembler* (správněji jazyk symbolických adres). Protože se většina projektů skládá z velkého množství takovýchto souborů, není účelné, aby se při změně jednoho souboru musela znova kompilovat celá aplikace. Řešením je rozdělení překladu z assembleru do absolutního strojového kódu cílového počítače na dvě části. Nejdříve jsou soubory s assemblerem (označované příponou **.s** nebo **.S**) přeloženy (programem **as**) do jednotlivých relokovatelných fragmentů strojového kódu (objektové soubory **.o**). Tyto soubory jsou pak linkerem spojovány do výsledné aplikace. Linker (**ld**) složí jednotlivé fragmenty, zjistí jejich umístění (adresy) v cílové aplikaci a nahradí symbolické odkazy binárními hodnotami. Protože velká část objektových souborů se opakuje ve všech aplikacích, jsou objektové soubory se standardními rutinami ukládány do knihoven (program **ar**) a linker později sám rozhodne, které rutiny jsou volány aplikačním programem a příslušné fragmenty i s jejich dalšími požadavky přidá do výsledného souboru.

3.3.1 Příprava instalace balíku BINUTILS

Balík se zdrojovými texty výše zmiňovaných programů je možné získat na velkém množství FTP serverů pod názvem **binutils-2.7.0.3.tar.gz** pro popísanou verzi. V naší oblasti je nejlépe přístupný na mirroru 'sunsite.mff.cuni.cz'. Ve většině případů je vždy výhodnější použít nejnovější verzi. V současné době je již k dispozici verze **2.8.1**.

Balík je komprimovaný (programem GZIP) UNIXový TAR archiv. Pro předpokládaný hostitelský systém **Linux** a požadované umístění zdrojových textů v adresáři `/usr/src/binutils-2.7.0.3` je nutné provést příkazy

```
cd /usr/src
tar -xzf binutils-2.7.0.3.tar.gz
```

Vznikne adresář *binutils-2.7.2.1* ve kterém jsou umístěny veškeré zdrojové texty kompilátoru programů. Pro uvažovanou cílovou platformu MC68332 je nutné nakonfigurovat kompilátor příkazy

```
cd /usr/src/binutils-2.7.0.3
./configure --host=i486-linux --target=m68k-coff \
--build=i486-linux --exec-prefix=/usr --prefix=/usr \
--with-shared
```

Poslední příkaz zabírající tři řádky provede vlastní konfiguraci programů. Některé údaje v něm nemusí být uvedeny, protože mohou být odvozeny implicitně, ale pro další výklad je vhodná kompletní podoba. Nejdůležitější je pro požadovanou cílovou platformu přepínač `-target=m68k-coff`.

V dalších odstavcích jsou vysvětleny jednotlivé aspekty konfigurace vedoucí k výše uvedenému příkazu. Vlastní kompilace je popsána v odstavci 3.3.7.

3.3.2 Kanonické jméno platformy

Pro popis platformy (prostředí) používají téměř veškeré GNU programy kanonického popisu. Ten podává veškerou informaci o typu procesoru, výrobci (tvůrci) operačního systému a o vlastním operačním systému.

CPU-COMPANY-SYSTEM

Ve velkém množství případů je možné některé části vypustit, protože jsou dostatečně určeny ostatními částmi. Například dále popisovaná verze překladače GCC rozpoznává více jak 200 různých kombinací těchto tří parametrů. Jedná se o následující procesory (CPU)

```
fx80
hppa1.0, hppa1.1
i370
i386, i486, i586
i860, i960
m68000, m68k
m88k
mips, mips64, mips64el, mips64orion, mips64orionel, mipsel
ns32k
pdp11
powerpc
powerpcle
```

```
pyramid
romp
rs6000
sparc, sparc64, sparclite
vax
we32k
xyes
```

Dále následují nejběžnější kombinace, které by se mohly čtenáři hodit. Hvězdičky nahrazují části názvů, které mají alternativy. V hranatých závorkách jsou místa, kde je možný výběr z více znaků.

```
i[345]86-*-bsd*   i[345]86-*-freebsd*   i[345]86-*-netbsd*
i[345]86-*-coff
i[345]86-*-linux
i[345]86-go32-msdos  i[345]86-*-go32
m68k-*-aout*
m68k-*-coff*
```

Novější verze umožňují i výběr přímo určený pro RT výkonné jádro **RTEMS**, který je téměř shodný s volbou *coff*.

```
m68k-*-rtems
```

Důležitou informací, která je získána ze jména operačního systému je použitý binární formát objektových a spustitelných souborů. Nejčastěji používané formáty budou popsány v následujících odstavcích.

3.3.3 Formáty objektových a spustitelných souborů

S vývojem prostředků pro kompilaci programů se vyvíjely i formáty, ve kterých jsou ukládány relokovatelné a absolutní objektové soubory a spustitelné programy. Postupně se zvyšovalo množství informací, vhodných pro optimalizaci, ladění, spojování objektových souborů a zavádění spustitelných souborů. V současné době jsou nejrozšířenější tři dále popsané formáty. Většina z nich je navržena tak, že je lze přizpůsobit pro uložení strojového kódu určeného téměř pro libovolný cílový procesor. Formáty se pro jednotlivé cílové procesory mírně liší, např. délkou ukládaných adres, pořadím bytů podle významnosti (LSB, MSB) atd. Například **BINUTILS** z obvyklé distribuce **Linuxu** podporují následující formáty.

elf32-i386 , a.out-i386-linux, coff-i386,
elf32-m68k, coff-m68k, ieee,
a.out-m68k-linux, a.out-sunos-big, elf32-sparc,
srec, symbolsrec, tekhex, binary, ihex, trad-core

Nejpoužívanější formáty jsou následující:

aout původní UNIXový formát

formát je velmi závislý na cílové platformě (systému i procesoru). Rozděluje se do velkého množství podskupin. Knihovny za běhu mapované do prostoru aplikačního programu musí být vždy mapovány na pevnou adresu přidělenou centrální autoritou (výrobcem systému), programy musí již při kompilaci znát napevno přidělené adresy vstupních bodů knihoven. Pro ukládání relokovatelných fragmentů se většinou na těchto platformách používá jiného proprietárního formátu.

coff Common Object File Format

dobře přenositelný formát vhodný především pro ukládání relokovatelných a absolutních objektových formátů téměř pro všechny druhy procesorů. Není nejvhodnější pro ukládání spustitelných dynamicky linkovaných programů. Nevýhodou je, že v ladící informaci o adresách přeložených řádek zdrojového kódu neuvádí ve jménu souboru i jméno adresáře. To se nepříznivě projeví při ladění velkých celků se zdrojovými texty umístěnými v celém stromu adresářů.

elf Executable and Linkable Format²

zatím nejmodernější formát vhodný pro relokovatelné objektové soubory, spustitelné programy, dynamicky linkované programy a knihovny. Umožňuje mapovat knihovny do libovolného neobsazeného místa běžícího programu. Program si může i zažádat o mapování rozšiřující i knihovny podle jména i za běhu. Knihovna obsahuje kód nezávislý na umístění (PIC - Position Independent Code). Vstupní body knihoven jsou linkovány i za běhu přes symbolická jména.

Je možné tak zvané Late Bindings. To znamená, že i vlastní hledání knihovny a vyčlenění prostoru je provedeno až při pokusu o volání některého vstupního bodu.

srec, ihex, tekhex, binary

formáty obsahující pouze obraz kódu a inicializovaných dat v paměti cílového procesoru. Jsou vhodné pro přenos dat například do programátoru paměti EPROM, nebo přímo do paměti cílového procesoru, který neobsahuje inteligentnější zavaděč programů.

Protože formátů je velké množství a operace s nimi v assembleru, linkeru a ostatních programech je možné zobecnit obsahuje balík BINUTILS knihovnu s obecným interfacem pro přístup k libovolnému formátu. Tato knihovna se jmenuje **BFD** (Binary Format Driver) a může být k jednotlivým programům balíku linkována dynamicky. V případě **Linuxu** je uložena například v souboru `/usr/lib/libbfd.so.2.7.0.3`. Při konfiguraci balíku **BINUTILS** je nutné rozhodnout, které formáty bude tato knihovna zpracovávat. Ve většině případech je vhodný formát nalezen přímo z kanonického jména platformy. Je však možné požadované formáty explicitně specifikovat při konfiguraci programem *configure* jak bude popsáno v části 3.3.6.

3.3.4 Platformy build, host a target

Pro konfiguraci a kompilaci je nutné rozhodnout na které platformě bude balík **BINUTILS** a ostatní části **GNU** vývojového prostředí kompilovány (build), později provozovány (host) - spouštěny programy *as*, *ld*, *gcc* atd. a pro kterou cílovou platformu mají být určeny výsledné objektové soubory a vytvořené spustitelné programy (target). Většinou je kompilována pouze nová verze kompilátoru a utilit na určité platformě, pak jsou všechna tato určení shodná a mohou být zjištěna automaticky. Ve zde popisovaném případě je požadavek vytvořit křížový překladač (cross compiler), který je kompilován na systému **Linux**, bude provozován na systému **Linux** a výsledné aplikace mají být určeny pro systém s procesorem **MC68332**. Tím je vysvětlena i kombinace build, host a target v sekci 3.3.1.

Další možností, kdy jsou všechny tři platformy rozdílné je takzvaný canadian cross. Teoreticky je například možné kompilovat **GNU** prostředí kompilovat na systému **Linux**, při čemž vzniklé programy budou používány pod systémem **DOS** pro tvorbu řídicích aplikací provozovaných na systému s **MC68332**.

Hlavičkové soubory pro jednotlivé platformy se jsou pojmenovány *mh-<platforma>.h* (pro hostitelskou platformu) a *mt-<platforma>.h* (pro

² Čtenář si pravděpodobně pomyslí, že se nejedná o nic zvláštního v porovnání s formátem dynamických knihoven DLL v systému Windows. Ovšem Windows natahují celý program i knihovny do paměti a provedou relokační změny přímo v kódu programu. Běžící program a knihovny musí být po dobu běhu celé v operační paměti nebo ve odkládacím souboru. Unix nemodifikuje výkonné (textové) části spustitelných souborů. Pouze při spuštění je vyhrazeno místo v adresovém prostoru procesu pro textové sekce programu a knihoven. Načtení vlastního kódu a alokace fyzické paměti se provádí až při prvním pokusu o přístup do konkrétní stránky vyhrazeného prostoru. Při nedostatku fyzické paměti je možné uvolnit některé textové stránky bez nutnosti jejich zápisu do odkládacího souboru, protože nejsou modifikované a je možné je v budoucnosti načíst znovu z mapovaného spustitelného souboru.

cílovou platformu) a nacházejí se v adresáři `/usr/src/binutils-2.7.0.3/config`.

3.3.5 Standardní umístění souborů

Protože jsou nástroje určeny především pro UNIXové systémy, počítají se standardním umístěním souborů ve stromové struktuře adresářů. UNIX předpokládá pouze jeden kořen všech adresářů. Další disky a síťové svazky se připojují pouze připojením (`mount`) k některému z již existujících adresářů. Zároveň je počítáno i s provázáním stromu více počítačů i s různou architekturou. Proto jsou jednotlivé programové balíky rozděleny na části závislé na hostitelské platformě, závislé na cílové platformě, sdílené v síti, privátní k danému počítači atd. Protože je standardní systém adresářů definován normou, znají programy přímo absolutní cesty k volaným programům.

Z výše uvedeného vyplývá, že je nutné všechny části vývojového prostředí nakonfigurovat pro shodnou strukturu adresářů. Zároveň pro budoucí kombinaci jednotlivých částí s originálními programy z distribuce **Linuxu** je výhodné dodržet konfiguraci použitou pro tyto programy.

Části balíku BINUTILS závislé na cílové platformě jsou umístěny do substromu pod adresářem `<prefix>/<target>`. V případě Linuxu je používán prefix pro základní vývojové prostředky `/usr`. Celá cesta pak vypadá `/usr/m68k-coff`. V podadresáři `bin` budou umístěny jednotlivé spustitelné programy (`as`, `gasp`, `ld`, `nm`, `ranlib`, `size` a `strip`). V podadresáři `lib` budou později umístěny knihovny funkcí pro cílovou platformu. Zároveň zde musí být adresář nebo odkaz na adresář se jménem `ldscripts`. V tomto adresáři hledá linker při spojování fragmentů kódu formát a popis uložení jednotlivých oblastí do výsledného souboru.

Ostatní programy nezávislé na cílové platformě mohou být umístěny do adresáře `<exec-prefix>/<bin>`. Není-li `exec-prefix` specifikován je použit `prefix`. Při instalaci těchto programů je nutné dát pozor, aby nebyla narušena funkčnost vývojových prostředků pro ostatní platformy. Pro instalaci v prostředí nevyužívajícím standardní systém adresářů je možné instalovat i programy i pro více platform do jednoho adresáře. Pak je možné programy odlišit pomocí předpony a přípony přímo ve jménu programu (`program-prefix` a `program-suffix`).

3.3.6 Parametry programu configure pro BINUTILS

Popis veškerých možných parametrů programu by vydal na samostatnou knihu, proto zde budou uvedeny jen nejdůležitější parametry, které má význam měnit.

Program je ve skutečnosti soubor skriptů pro interpret Berkley Shell

systému UNIX. Spouští se příkazem `./configure3`, který je následován požadovanými parametry. Pro neuvedené parametry jsou vždy nalezeny optimální implicitní hodnoty. V následujícím textu jsou uvedeny v hranatých závorkách.

- `-help` vypíše nápovědu [neaktivní]
- `-build=BUILD` platforma, pro kompilaci prostředků [BUILD=HOST]
- `-host=HOST` provozní platforma, na které budou prostředky provozovány [zjištěna přes `config.guess`]
- `-target=TARGET` cílová platforma prostředků, vývojového prostředí [TARGET=HOST]
- `-prefix=MYDIR` základní adresář, kam bude provedena instalace prostředků [usr/local]
mají-li být prostředky kombinovány s ostatními vývojovými prostředky i pro jiné architektury, je nutné aby všechny měly MYDIR shodné. Standardní umístění pod systémem **Linux** je pod adresářem /usr.
- `-exec-prefix=MYDIR` programy nezávislé na cílové platformě MYDIR [usr/local]
použito např. v případě nutnosti rozlišení programů pro jednotlivé hostitelské platformy.
- `-norecursion` provést konfiguraci pouze aktuálního adresáře [recurse]
- `-program-prefix=FOO` přidat FOO před jména instalovaných programů [""]
pro cross kompilátory je v novějších verzích přidán implicitně *prefix* podle jména cílové platformy, pro uvažovanou konfiguraci `m68k-coff`.
- `-program-suffix=FOO` přidat FOO za jména instalovaných programů [""]
- `-srcdir=DIR` cesta k zdrojovým textům balíku [. nebo ..]
výhodné například při kompilaci z read-only zdroje, např. CD-ROM
- `-tmpdir=TMPDIR` používat adresář TMPDIR pro dočasné soubory [/tmp]

³ znaky tečka a lomeno “./” před jméno programu jsou nutné, protože většina instalací UNIXu má nastaveno v cestě pro hledání spustitelných souborů nejdříve standardní adresáře a až na konci je uveden aktuální adresář. Přidáním části cesty k programu se zaručí, že nebude spuštěn jiný program, než právě ten v aktuálním adresáři.

- infodir=DIR hypertextovou dokumentaci info do adresáře DIR [PREFIX/info]
- mandir=DIR klasickou dokumentaci man dokumentaci do DIR [PREFIX/man]
- nfp konfigurace pro softwarovou desetinnou aritmetiku [hard float]
- with-FOO, -with-FOO=BAR kompilovat s balíkem FOO (parametrem
 BAR)
- without-FOO balík FOO není k dispozici pro hostitelskou platformu
- enable-FOO, -enable-FOO=BAR kompilovat s vlastností FOO (parametrem
 BAR)
- disable-FOO nakázat vlastnost FOO
- enable-targets povolit podporu pro jednotlivé cílové platformy
 jedná se o podporu formátů a procesorů v knihovně libbfd, například konfigurace pro MC68832 a i386 **Linux** je
 -enable-targets=m68k-coff,i386-linux
- enable-shared kompilovat se sdílenou knihovnou BFD
- enable-commonbfdlib vytvořit sdílené knihovny BFD/opcodes/libiberty
- with-mmap používat mmap pro vstupní soubory v BFD

3.3.7 BINUTILS - vlastní kompilace a instalace

Tato část je relativně jednoduchá. Kompilace je spuštěna příkazem

```
cd /usr/src/binutils-2.7.0.3
make
```

Instalaci je možno provést příkazem *make install*. Pro kontrolu kolizí je však výhodnější pouze zjistit co je třeba kam zkopírovat a zkontrolovat případné kolize s původními programy. Zjištění příkazů, které by byly vykonány se provede přidáním přepínače *-n*. Pak příkaz i se zachycením výstupu vypadá následovně

```
make install -n | less
```

Je-li použito sdílené knihovny BFD s vhodnou kombinací používaných platform, jsou teoreticky pro přidání další platformy specifické pouze soubory *as* a *ld*. Vše ostatní může být nahrazeno symbolickými linkami.

3.3.8 GNU Assembler “as”

Tento program je obvykle volán pouze přes integrovaný překladač **gcc**. Přesto je vhodné pro některé účely znát jeho použití. Pro příslušnou cílovou platformu je nutné volat správný program (např. `/usr/m68k-coff/bin/as`). Stručnou nápovědu je možné získat voláním programu s parametrem `-help`. Pro kompletní popis je nutné použít dokumentaci ve formátu info (prohlížet např. programem `info` nebo v editorech `emacs` a `jed` klávesami `<ctrl+h>` `<i>`). Stručně : na řádce se bez přepínače zadávají vstupní soubory, za přepínač `-o` se zadává jméno výstupního souboru. Pro zde uvažovanou platformu jsou důležité přepínače mezi variantami procesoru, které povolují překlad nových instrukcí oproti MC68000. Jedná se o přepínače typu `-m68332`, `-m68020`, `-m68881` (pro příslušnou FPU) atd. Tvorbu kódu nezávislého na umístění (např. sdílené knihovny ELF viz 3.3.3) se vyžádá přepínačem `-pic`. Pro překlad souborů s originální (Motorola) syntaxí instrukcí (nikoli AT&T a UNIX) je možno použít přepínač `-M`.

3.3.9 GNU Linker “ld”

Opět se jedná o program závislý na platformě (např. `/usr/m68k-coff/bin/ld`). Nápověda opět `-help` nebo ve formátu info. Bez přepínače se zadávají jednotlivé vstupní soubory (`*.o` a `lib*`). Implicitní jméno vytvořeného spustitelného souboru `a.out` lze změnit přepínačem `-o`. Knihovny lze uvádět také za přepínačem `-l` (pak se uvádí bez přípon a počátečního `lib`), které jsou hledány v adresářích definovaných za přepínači `-L`. Konfigurace cílového paměťového prostoru (počátek, délka, sekce, RAM, ROM atd.) je definována ve skriptu v adresáři `ldscripts` (např. `m68kcoff.x`). Jiný než implicitní skript lze vybrat přepínačem `-T` a je hledán v knihovních adresářích. Adresář `ldscripts` obsahuje pro každou platformu více skriptů, které se liší příponami:

- *.**x** normální spustitelný soubor (program)
- *.**xr** pro relokovatelný výstup z linkeru s přepínačem `-r`, výstupní formát musí podporovat relokovatelný kód.
- *.**xu** relokovatelný kód bez vytvoření konstruktorů, přepínač `-Ur`
- *.**xn** sekce `.data` a `.text` mohou být kombinovány, přepínač `-n`
- *.**xbn** totéž, ale s přepínačem `-N`
- *.**xs** pro kompilaci sdílené knihovny pokud je přepínač `-shared`

Výstupní formát lze změnit přepínačem *-offormat* . Je-li kompilována sdílená knihovna musí být nastaveno *-shared* a může být definováno *-soname* (interní jméno knihovny), při kompilaci programů je možné linkovat proti statickým knihovnám *-Bstatic* (knihovny **lib*.a**) nebo dynamickým *-Bdynamic* (**lib*.so**).

Pro embeded aplikace je nutné vytvořit správný ldscript a nebo lze určit počátek paměti EPROM, RAM a oblasti zásobníku následujícími přepínači *-Ttext 0x10000 -Tdata 0x18000 -Tbss 0x1C000*.

3.3.10 Konvertor “objcopy”

Objcopy je velmi jednoduchý program, který otevře vstup a výstup přes BFD ve specifikovaném formátu a překopíruje veškerou informaci, kterou je možné do výstupního formátu uložit. Například vytvoření S-record souboru pro download programu do testovací desky s lokálním monitorem schopným tento formát přijmout

```
objcopy --input-target=m68k-coff \  
--output-target=srec -S a.out a.srec
```

Další užitečné přepínače *-g* a *-S* pro potlačení ladících informací. *-R* pro nekopírování určité sekce.

Další příklad je vytvoření obrazu paměti v Intel HEX formátu pro naprogramování do paměti EPROM

```
objcopy --input-target=m68k-coff \  
--output-target=ihex a.out a.hex
```

Přepínače *-byte* a *-interleave* umožňují řešit rozložení dat do více paralelně řazených pamětí. Přepínače *-set-start* a *-adjust-start* umožňují vybrat jen určitou část z dat se správným posunutím.

Program podporuje libovolnou konverzi formátů, můžeme však narazit na ztrátu některých informací, jiné konvence jmen (např. počáteční znak “_” před jménem lze programem objcopy potlačit). Pro relokovatelné soubory může dojít k ztrátě relokačních informací.

3.3.11 BINUTILS - další programy

objdump vypočítá obsah souboru s fragmentem kódu nebo spustitelného programu

např. *-source* vypíše všechny sekce s kódem ve formátu listingu assembleru proložené řádkami zdrojového kódu (jsou-li zdrojové soubory k dispozici)

nm výpis symbolických jmen použitých v daném binárním souboru

size výpis velikost jednotlivých sekcí programu

ar program pro správu knihoven

ranlib vytvoří index symbolických jmen v knihovně pro zvýšení rychlosti při sestavování programů

strip vymazání ladících informací

gasp GNU assembler macro preprocessor

3.4 GCC překladač jazyka “C”

V současné době je základním jazykem pro psaní systémových programů jazyk “C”, jehož dostupnost na téměř všech platformách umožňuje i přenositelnost aplikací. Překladač **GCC** je navržen tak, aby umožňoval i snadnou svojí vlastní přenositelnost i na systémy, na kterých zatím nebyl provozován a i pro nové systémy, pro které zatím jiný překladač neexistuje. Tato vlastnost je velmi výhodná i pro vývoj řídicích aplikací, kde se implementace samotného kompilátoru **GCC** na cílové platformě nepředpokládá.

Kvalita výsledného kódu v assembleru je přinejmenším srovnatelná s většinou komerčních překladačů. Překladač je postupně optimalizován a rozšiřován o další cílové (target) a hostitelské (host) platformy.

V dalším textu bude uvažována verze **2.7.2.1**, operační systém hostitelského počítače **Linux** a cílová platforma **MC68332**. V současné době již existuje verze **2.8.0** a nová vývojová větev **EGCS**. Verze **EGCS** zahrnuje množství vylepšení a dalších optimalizací. Předpokládá se, že po důkladném otestování budou postupně vylepšení přejímána do stabilní vývojové větve **GCC**.

3.4.1 GCC - příprava instalace crosscompileru

Křížový překladač je taková konfigurace, kdy cílová platforma je jiná než hostitelská platforma vlastního překladače. Balík se zdrojovými texty kompilátoru je možné získat na velkém množství FTP serverů pod názvem **gcc-2.7.2.1.tar.gz** pro popisovanou verzi. V naší oblasti je opět nejlépe přístupný na mirroru ‘**sunsite.mff.cuni.cz**’.

Balík je komprimovaný (programem GZIP) UNIXový TAR archiv. Pro předpokládaný hostitelský systém Linux a požadované umístění zdrojových textů v adresáři */usr/src/gcc-2.7.2.1* je nutné provést příkazy

```
cd /usr/src
tar -xzf gcc-2.7.2.1.tar.gz
```

Vznikne adresář gcc-2.7.2.1 ve kterém jsou umístěny veškeré zdrojové texty kompilátoru GCC. Pro uvažovanou cílovou platformu MC68332 je nutné nakonfigurovat kompilátor příkazy

```
cd /usr/src/gcc-2.7.2.1
./configure --host=i486-linux --target=m68k-coff \
--build=i486-linux --with-gnu-ld --with-gnu-as --nfp
```

Druhý z příkazů je dlouhý přes dvě řádky. Některé údaje v něm nemusí být uvedeny, protože mohou být odvozeny implicitně. Nejdůležitější je pro požadovanou cílovou platformu opět přepínač `-target=m68k-coff`.

3.4.2 Parametry programu configure pro GCC

Program je opět soubor skriptů pro interpret Berkley Shell systému UNIX. Spouští se příkazem `./configure`, který je následován požadovanými parametry. Způsob zadávání jmen platform je shodný s balíkem **BINUTILS** viz 3.3.4. Pro neuvedené parametry jsou vždy nalezeny optimální implicitní hodnoty. V následujícím textu jsou uvedeny v hranatých závorkách.

- `-help` vypíše nápovědu [neaktivní]
- `-build=BUILD` platforma, pro kompilaci prostředků [BUILD=HOST]
- `-host=HOST` provozní platforma, na které budou prostředky provozovány [zjištěna přes config.guess]
- `-target=TARGET` cílová platforma prostředků, vývojového prostředí [TARGET=HOST]
- `-prefix=MYDIR` základní adresář, kam bude provedena instalace prostředků [/usr/local]
mají-li být prostředky kombinovány s ostatními vývojovými prostředky i pro jiné architektury, je nutné aby všechny měly MYDIR shodné. Standardní umístění pod systémem **Linux** je pod adresářem /usr.
- `-local-prefix=MYDIR` kde se nalézá adresář include s lokálními hlavičkovými soubory [/usr/local]
je nutné, aby podadresář include neobsahoval systémové hlavičkové soubory, proto MYDIR nesmí být /usr

- exec-prefix=MYDIR programy nezávislé na cílové platformě MYDIR [/usr/local]
použito např. v případě nutnosti rozlišení programů pro jednotlivé hostitelské platformy.
- norecursion provést konfiguraci pouze aktuálního adresáře [recurse]
- program-prefix=FOO přidat FOO před jména instalovaných programů [""]
opět pro nové verze GCC platí, že pro crosscompiler je automaticky předřazeno jméno cílové platformy
- program-suffix=FOO přidat FOO za jména instalovaných programů [""]
- srcdir=DIR cesta k zdrojovým textům balíku [. nebo ..]
výhodné například při kompilaci z read-only zdroje, např. CD-ROM
- tmpdir=TMPDIR používat adresář TMPDIR pro dočasné soubory [/tmp]
- infodir=DIR hypertextovou dokumentaci info do adresáře DIR [PREFIX/info]
- mandir=DIR klasickou dokumentaci man dokumentaci do DIR [PREFIX/man]
- nfp konfigurace pro softwarovou desetinnou aritmetiku [hard float]
- with-FOO, -with-FOO=BAR kompilovat s balíkem FOO (parametrem BAR)
- without-FOO balík FOO není k dispozici pro hostitelskou platformu
- enable-FOO, -enable-FOO=BAR kompilovat s vlastností FOO (parametrem BAR)
- disable-FOO zakázat vlastnost FOO
- with-gnu-as GCC bude využívat GNU assembler
má smysl pouze pro systémy, pro které není GNU toolchain základním vývojovým prostředím. GNU assembler musí být již nainstalován z balíku BINUTILS.
- with-gnu-ld totéž pro GNU linker

3.4.3 Soubory používané pro konfiguraci GCC

Při kompilaci je zavedena cesta k hlavičkovým souborům do adresáře *config*, ve kterém je uložena většina informací o specifických platformách. V základním adresáři se zdrojovými texty GCC jsou konfiguračním skriptem vytvořeny odkazy na tyto soubory.

Při konfiguraci je vytvořen v základním adresáři zdrojového textu GCC soubor *config.h*, který přes direktivu *#define* zpřístupňuje soubor definic pro hostitelskou platformu *<CPU>/xm-<SYSTEM>.h* z adresáře *config*. Pro zde uvažovanou konfiguraci to bude soubor *i386/xm-linux.h*. Tento soubor dále zpřístupňuje informace ze souborů obsahujících informace o procesoru *<CPU>/xm-<CPU>.h* a obecné informace o hostitelském systému *xm-<SYSTEM>.h*, v předkládaném případě *i386/xm-i386.h* a *xm-linux.h*. Soubor s definicemi pro hostitelský procesor obsahuje další odkaz na informace o cílové platformě kompilátoru *tm.h*.

Cílová platforma je popsána podobným postupem jako platforma hostitelská. Soubor *tm.h* v základním adresáři se odkazuje na soubor *<CPU>/<SYSTEM>.h*. V popisované konfiguraci *m68k/m68k-coff.h*, což je způsobeno tím, že není vytvářen kompilátor pro konkrétní cílový systém, ale pro kombinaci procesoru a jednoho z možných formátů objektových souborů. Tento soubor se většinou odkazuje na několik dalších souborů obsahujících konkrétní informace o cílovém procesoru *<CPU>/<CPU>.h*, obecné informace o cílovém systému *<SYSTEM>.h* a o použitém formátu objektových souborů.

Soubor *tconfig.h* obsahuje konfiguraci použitou již zkompilevaným kompilátorem při kompilaci knihoven. V uvažovaném případě je jeho obsah shodný s *tm.h*.

Při konfiguraci je také ze souboru *Makefile.in* vytvořen soubor *Makefile*, do kterého mohou být přidány části závislé na hostitelské a cílové platformě uložené v adresářích pro příslušné procesory a nesoucí jména *x-<HOST>* a *t-<TARGET>*.

V adresáři pro cílový procesor se nachází také vlastní popis vlastností a instrukcí cílového procesoru v souboru *<CPU>.md*. Tento popis je doplněn algoritmy pro některé důležité operace (např. generování vstupního a výstupního bloku podprogramu) v souboru *<CPU>.c*.

3.4.4 GCC - vlastní kompilace

V nejjednodušším případě stačí spustit kompilaci ze základního adresáře balíku GCC příkazem *make*. Pokud jsou požadovány jen některé z variant kompilátoru je možné specifikovat v příkazu

```
make LANGUAGES="<LIST>"
```

Položka <LIST> může obsahovat libovolnou kombinaci následujících variant oddělených mezerami

c kompilátor standardního jazyka “C”

c++ kompilátor jazyka C++

objective-c kompilátor verze objective-C

proto povoluje kompilace programů *protoize* a *unprotoize*

tyto programy slouží pro převod zdrojových textů z tvaru Kerningham&Ritchie do tvaru ANSI a naopak

Je-li kompilátor kompilován na cílové platformě je vhodné použít dalších kompilací, k vyloučení vlivu předcházejících verzí (nebo jiného než GNU kompilátoru) na výsledný kompilátor. To se provádí opakováním kompilace kompilátoru kompilátorem vzniklým v předchozí etapě. Zároveň je možné postupně přidat stupeň optimalizace kódu výsledného kompilátoru. Další kompilace se provede příkazem

```
make stage1
make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O2"
```

výsledky minulé etapy jsou uloženy do adresáře *stage1* a jsou použity pro další kompilaci. Pro platformy bez hardwarové podpory výpočtů v plovoucí řádové čárce může být nutné specifikovat při přechodu z jiného než GNU kompilátoru emulaci těchto výpočtů modifikací druhého příkazu

```
make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O2 \
-msoft-float"
```

Pro získání jistoty o kvalitě výsledného kompilátoru je doporučeno provést třetí kompilaci

```
make stage2
make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2"
```

a zkontrolovat, jestli se výsledné objektové soubory shodují s těmi, které vznikly v předchozí etapě a jsou uloženy v adresáři *stage2*. Tato komparace nemusí být triviální na platformách, které ukládají do objektových souborů datum kompilace. Pak je nutné provést komparaci následujícím pomalejším způsobem

```
make compare
```

Libovolná zjištěná diference je chybou a měla by být oznámena vývojářům překladače GCC.

V novějších verzích lze provést několikaúrovňový překlad s kontrolou činnosti výsledného kompilátoru jediným požadovaným cílem programu *make bootstrap*.

3.4.5 GCC - kompilace crosscompileru

GCC je schopno kompilovat zdrojový kód pro jinou cílovou platformu než je platforma, na které je provozováno. Tomuto způsobu práce se říká crosscompilace a je potřebný pro cílové platformy, pro které ještě neexistuje kompilátor jazyka "C", nebo na kterých nemůže být kompilátor provozován. Příkladem jsou řídicí a regulační jednotky.

S kompilací crosscompileru mohou nastat problémy, má-li cílová platforma jinou délku operačního slova nebo když popis cílové platformy správně nevyužívá možnosti emulace plovoucí řádové čárky při kompilaci.

Konfigurace se provede pro uvažovanou cílovou platformu *m68k-coff* například příkazem (viz 3.4.2)

```
./configure --target=m68k-coff
```

V době kompilace by již měly být nainstalovány programy z balíku BINUTILS (viz 3.3) v adresáři *<prefix>/<target>/bin*. Pro uvažovaný případ */usr/m68k-coff*.

Jsou-li k dispozici knihovny a hlavičkové soubory pro cílovou platformu, mohou být umístěny do adresářů *<prefix>/<target>/lib* respektive *<prefix>/<target>/include*. Důležité jsou také soubory pro spuštění programu **crt*.o*.

Problémy také mohou nastat s knihovnou emulací některých operací (např. operace v plovoucí řádové čarce nebo operace celočíselného násobení a dělení) *libgcc.a* pro některé cílové platformy. Ve většině případů však může být tato knihovna prázdná nebo vybudována ze zdrojových textů dodávaných spolu s GCC.

Spuštění vlastní kompilace se provede stejně jako při kompilaci obyčejného kompilátoru

```
make LANGUAGES="<LIST>"
```

Nejsou-li k dispozici hlavičkové soubory pro tvorbu *libgcc.a*, kompilace se v určité fázi přeruší. Některé starší verze GCC také předpokládají již existenci alespoň prázdné knihovny *libgcc.a*. Pro další pokračování je nutné vytvořit v adresáři *<prefix>/<target>/lib* alespoň prázdnou knihovnu příkazem

```
ar -cd libgcc.a dummy
```

Po opětovném spuštění příkazem *make* kompilace pokračuje a v dalších fázích doplní knihovnu *libgcc.a*. Úplnost knihovny zkontroluje kompilací testovacího programu *libgcc1-test*. Dojde-li při jeho linkování k chybě, znamená to, že ne všechny potřebné rutiny byly do knihovny doplněny a náprava je možná pouze jejím manuálním doplněním.

Z hlavičkových souborů je k dokončení kompilace nutný soubor *float.h*. Tento soubor může být získán po zkompilování a spuštění testovacího programu *enquire* na cílové platformě. Tento postup není většinou pro řídicí systémy možný, pak je nutné použít již předdefinovaný hlavičkový soubor *float.h*. Tento soubor použitelný pro většinu cílových platform je k dispozici z cross-gcc distribuce firmy CYGNUS.

Při kompilaci crosscompileru není možné provést následující etapy kompilace, protože již zkompilovaný kompilátor neumí kompilovat pro hostitelskou platformu. Po správné instalaci crosscompileru a nové konfiguraci GCC je však možné zkompilovat kompilátor určený pro běh na cílové platformě.

3.4.6 GCC - instalace crosscompileru

Nejjednodušší je spustit instalaci příkazem

```
make install
```

Je však vhodné vědět, do kterých adresářů jsou uloženy jednotlivé programy a soubory kompilátoru. Základní adresář, do kterého je GCC instalováno, je adresář `<prefix>/lib/gcc-lib/<target>/<version>`. V uvažované konfiguraci `/usr/lib/gcc-lib/m68k-coff/2.7.2.1`. V tomto adresáři se budou nacházet následující soubory. Programy **gcc** a **g++** s případnou předponou jsou uloženy do adresáře `<exec-prefix>` pro binární programy.

cpp makro preprocesor jazyka "C"

cc1 vlastní kompilátor jazyka "C"

cc1plus kompilátor jazyka C++

cc1obj kompilátor jazyka objective-C

gcc-cross obálka pro volání všech verzí crosscompileru

v případě normálního (nativního) kompilátoru se program jmenuje **gcc**, v novějších verzích je pro crosscompiler použito jméno `<target>-gcc`, například **m68k-coff-gcc**, volba záleží na volbě parametru *program-prefix* při konfiguraci

g++-cross obálka pro přímou kompilaci zdrojových textů v jazyce C++

specs popis akcí spouštěných programem **gcc-cross**

tento soubor obsahuje vztahy mezi přepínači pro program **gcc-cross** a ostatní jím volané programy (**cc1**, **cc1plus**, **as**, **ld**). V souboru jsou obsaženy i informace o knihovnách, které je nutné přidat pro softwarovou emulaci operací s plovoucí řádovou čárkou, knihoven a startovacích souborů pro umožnění ladění.

libgcc.a knihovna s emulačními a pomocnými funkcemi

tyto funkce jsou volány z přeloženého programu v případech, kdy přímé vložení sekvence instrukcí není možné nebo celková délka je příliš velká (např. emulace operací v plovoucí řádové čárce). Tato knihovna může být uložena v několika variantách pro různé kombinace přepínačů *-m* zpřesňujících variantu cílové architektury. Varianty jsou pak uloženy v podadresářích pojmenovaných jmény těchto přepínačů.

libobjc.a knihovna pomocných funkcí pro kompilátor objective-C

m<variant> adresáře obsahující varianty knihovny **libgcc.a**

include hlavičkové soubory pro některé funkce závislé i na kompilátoru

například operace pro práci s proměnným počtem vstupních argumentů funkcí

4 BDM vývojové rozhraní a GDB

konec textu

Reference

- [1] M68300 Family MC68332, MOTOROLA, INC. 1995
<http://www.mot-sps.com/>
- [2] CPU32 REFERENCE MANUAL MOTOROLA, INC., 1990, 1996
- [3] TPU TIME PROCESSOR UNIT REFERENCE MANUAL, MOTOROLA, INC., 1996
- [4] MCF5206 USER'S MANUAL MOTOROLA, INC., pre-final version
- [5] RCPURISC CENTRAL PROCESSING UNIT REFERENCE MANUAL, MOTOROLA, INC. 1994, 1996
- [6] Great Microprocessors of the Past and Present (V 10.1.1),
John Bayko (Tau), March 1998,
<http://www.cs.uregina.ca/~bayko/>
- [7] Frequently Asked Questions FAQ, For the Internet USENET newsgroup:
comp.sys.m68k,
Robert Boys, Ontario, CANADA, August 24, 1995, Version 19,
<http://www.ee.ualberta.ca/archive/m68kfaq.html>
- [8] Debugging with GDB The GNU Source-Level Debugger Fifth Edition,
for GDB version 4.17 , Richard M. Stallman and Roland H. Pesch, April
1998
Copyright (C) 1988-1998 Free Software Foundation, Inc
Free Software Foundation 59 Temple Place - Suite 330, Boston, MA
02111-1307 USA
ISBN 1-882114-11-6

Obsah

1	Popis mikrokontroléru	1
1.1	Přehled vlastností jednotlivých modulů	2
1.2	Stručný popis CPU32	4
1.2.1	Registry a režimy procesoru	6
1.2.2	Adresace - konvence	7
1.2.3	Režimy adresace	8
1.2.4	Instrukční soubor	9
1.2.5	Výkonnější následníci CPU32	9
1.3	Integrační modul (SIM)	10
1.4	Časovací koprocessor (TPU)	11
1.5	Komunikační modul (QSPI)	12
2	Přehled vývojových prostředků	12
2.1	Hardwarové prostředky	12
2.2	Prostředky pro vývoj software	13
2.3	Ladění aplikací	14
3	GNU Tool Chain	15
3.1	Projekt GNU a FSF Inc	15
3.2	Programy nutné pro přípravu vývojového prostředí	16
3.3	BINUTILS assembler, linker a další utility	17
3.3.1	Příprava instalace balíku BINUTILS	17
3.3.2	Kanonické jméno platformy	18
3.3.3	Formáty objektových a spustitelných souborů	19
3.3.4	Platformy build, host a target	21
3.3.5	Standardní umístění souborů	22
3.3.6	Parametry programu configure pro BINUTILS	22
3.3.7	BINUTILS - vlastní kompilace a instalace	24
3.3.8	GNU Assembler “as”	25
3.3.9	GNU Linker “ld”	25
3.3.10	Konvertor “objcopy”	26
3.3.11	BINUTILS - další programy	26
3.4	GCC překladač jazyka “C”	27
3.4.1	GCC - příprava instalace crosscompileru	27
3.4.2	Parametry programu configure pro GCC	28
3.4.3	Soubory používané pro konfiguraci GCC	30
3.4.4	GCC - vlastní kompilace	30
3.4.5	GCC - kompilace crosscompileru	32
3.4.6	GCC - instalace crosscompileru	33

Seznam obrázků

1	Funkční bloky 68332	3
2	Uživatelský model CPU32	5
3	Systémový model CPU32	5
4	Stavový registr	7
5	Časovací koprocessor TPU	11
6	Externí emulátor	13
7	Interní BDM emulátor	14