Moderní mikrokontroléry a využití vývojového prostředí GNU

Pavel Píša (pisa@cmp.felk.cvut.cz)

12. listopadu 2001

1 Úvod

Tento dokument se snaží čtenáři přiblížit architektury a možnosti moderních mikrokontrolérů a mikroprocesorů používaných pro řídící aplikace, které patří do střední a vyšší výkonnostní třídy. Vývoj mikroprocesorových jader používaných pro řízení sleduje ve většině případů s určitým časovým odstupem vývoj procesorů pro mikropočítače a pracovní stanice. Proto bude v první části stručně přiblížena obecná historie a trendy vývoje mikroprocesorů. Dále bude podrobněji popsána architekturu 32-bitových mikrokontrolérů od firmy Motorola vycházejících z rodiny mikroprocesorů 68000, které jsou pro svojí eleganci velmi často nasazovány ve složitějších systémech. V dnešní době je již tato architektura nedostatečná pro extrémně náročné aplikace (například zpracování video signálu nebo řízení vysokorychlostních komunikací). Proto budou uvedeny stručné informace o dalších podobných mikrokontrolérech a o jejich výkonných následnících. Nejnovější verzi tohoto dokumentu lze nalézt v elektronické podobě na adrese `http://cmp.felk.cvut.cz/~pisa/`.

2 Stručný přehled vývoje architektur mikroprocesorů

Prvním integrovaným mikroprocesorem byl obvod 4004 vyvinutý firmou Intel v roce 1971. Jednalo se o čtyřbitový mikroprocesor s adresační schopností 1kB dat a 4kB kódu. První mikroprocesory využívaly akumulátorovou architekturu, to znamená, že veškeré aritmetické operace ukládaly výsledek do jednoho registru - akumulátoru. Některé neuměly nepřímé adresování, jiným zase chyběly možnosti absolutní adresace a využívaly pouze nepřímé adresování přes specializované indexové a adresní registry.

Osmibitový mikroprocesor Intel 8080 (1974) se sedmi registry, ukazatelem zásobníku a čítačem programu již umožňoval využití dvojic registrů pro 16-bitovou nepřímou adresaci 64kB společné paměti dat a programu. Protože rychlost pamětí se v tomto čase zvýšila natolik, že přestala být omezujícím faktorem pro tehdejší procesory, vyvinuly některé firmy mikroprocesory s velmi malým počtem registrů, například 6800 od Motoroly a 6502 od NMOS Technologies (1975). Tyto obvody obsahovaly pouze ukazatel zásobníku, akumulátor a jeden indexový registr. Instrukční soubor nemusel obsahovat množství kombinací

kódů pro jednotlivé registry a volný kódový prostor byl vyplněn kombinacemni široké nabídky adresních režimů. Jako zajímavost lze uvést pozdější mikroprocesor TMS 9900 firmy Texas Instruments, který neměl žádný uživatelský registr ani ukazatel instrukcí integrovaný na čipu, ale oblast registrů v paměti byla určena ukazatelem na pracovní oblast. To umožňovalo velmi rychlé přepínání úloh a podobné řešení využívají i dnešní transputery.

Pozdější zlepšení technologie výroby a architektury mikroprocesorů vedlo k tomu, že mikroprocesory předstihly rychlost pamětí a dopravení kódů instrukcí a dat do a z mikroprocesoru se stalo jedním z hlavních omezení výpočetního výkonu procesorových systémů. Proto byly vyvíjeny mikroprocesory s větším množstvím registrů. Zároveň používání kompilátorů vedlo ke snaze o ortogonalitu architektur (co nejméně rozdílů mezi jednotlivými registry) a ke komplexním adresačním módům, které pokud to bylo možné co nejvíce odpovídaly konstrukcím programovacích jazyků (v té době především jazyku "C"). Do tohoto období spadá i vývoj mikroprocesoru Motorola 68000 (1979) pojmenovaný podle počtu integrovaných tranzistorů na čipu. Cenou za kompaktní instrukční soubor a složité adresní módy byla nutnost využít pro interpretaci instrukcí mikrořadiče s mikrokódem. To sice zjednodušilo a zrychlilo návrh mikroprocesoru, ale zároveň vedlo k zvýšení počtu použitých tranzistorů a k pomalejšímu výkonu instrukcí. Pozdější verze rozdělily výkon instrukcí nejdříve mezi dvě jednotky (jednotka pro komunikaci se sběrnicí a výkonná jednotka v procesorech 68010, mikrokontrolérech 683xx). Ještě později byl výkon instrukcí rozdělen do více překrývajících se fází (tři u procesoru 68020) až nakonec v procesorech 68060 při desetifázovém zpracování (10-stage pipeline) docházelo při třetí fázi k dekódování instrukcí do vnitřního tvaru, který již nebylo nutno dále interpretovat s využitím mikrokódu. Několik výkonných jednotek mohlo zpracovávat instrukce ve vnitřním tvaru paralelně (superskalární architektura). Od verze 68030 byla na čip procesoru integrována i jednotka správy paměti a od verze 68040 i koprocesor pro operace v plovoucí řádové čárce.

Zajímavostí je, že firma Zylog známá svým procesorem Z-80 vyvinula již v roce 1986 šestnáctibitový a třicetidvoubitový mikroprocesor Z-8000 a Z-80000, které díky přímému šestifázovému zpracovaní instrukcí nemusely využívat mikrokód. Procesor vystačil pouze s 18000 tranzistory a technologicky o pět let předběhl vývoj mikroprocesorů Intel a Motorola. Komerčně se ovšem neprosadil.

Velmi podobným vývojem jako mikroprocesory Motorola prošly i mikroprocesory firmy Intel, u kterých ovšem díky použití množství prefixů, módů a nesystematičností v instrukčním souboru byla cesta k vyloučení mikrokódu mnohem obtížnější. Firma Intel na rozdíl od Motoroly při vývoji šestnáctibitového procesoru 8086 (1978) nerozšířila adresové registry tak aby pokryly celý adresní prostor 1 MB, ale zvolila adresaci s využitím šestnáctibitových segmentových registrů násobených při výpočtu fyzické adresy šestnácti. Šestnáctibitová délka registrů byla zachována i u mikroprocesoru 80286, který již podporoval 16 MB adresní prostor a lokální a globální tabulku deskriptorů pro umístění a ochranu segmentů a procesů v paměti. Až registry procesoru 80386 byly rozšířeny na 32 bitů. Segmentovací logika byla zachována a dále byla přidána jednotka správy stránkování paměti. Procesor 80486 již implementoval většinu často prováděných instrukcí bez použití mikrokódu a měl integrovaný matematický koprocesor. Procesory Pentium a výše jsou již superskalární a mikrokód slouží pouze pro složité operace pro správu a přepínání procesů.

I přes výkonný kompaktní instrukční soubor byla rychlost pamětí stále nedostatečná. Tento problém byl vcelku uspokojivě vyřešen využitím vyrovnávacích pamětí cache. Některé procesory se též snažily o zrychlení a zjednodušení přístupu k zásobníku, který se kromě ukládání návratových adres stal prostorem pro předávání parametrů podprogramům a pro ukládání lokálních dat. Tyto mikroprocesory zavedly takzvaná registrová okna, která vlastně tvořila jakousi cache nejvyšší části zásobníku. Například mikroprocesory SPARC obsahují 8 obvyklých registrů a 48 přístupných registrů reprezentujících část zásobníku. Dalších více než 100 registrů slouží pro schraňování odložených skupin po 16 registrech. Po naplnění všech těchto registrů dochází k automatickému přesunu dat do vnější paměti. Data jsou zpět do registrů načtena při snížení úrovně vrcholu zásobníku.

Další možností je zrychlit volání leaf-node (nevolají další podprogramy) podprogramů náhradou klasického průběhu volání s uložením návratové adresy na vrchol zásobníku. Toho je dosaženo přesunem adresy následující instrukce za instrukcí volání do specializovaného návratového registru, který je použit instrukcí návratu z podprogramu. Podprogramy, které volají další vnořené funkce, musí však zajistit uchování stavu návratového registru. Tento princip je použit například u mikroprocesorů PowerPC.

Přidání vyrovnávacích pamětí cache a další výše uvedená zlepšení způsobila, že se problémovým místem opět stal vlastní výkon instrukcí a to hned z několika důvodů. Více fázové zpracování sice výrazně zrychluje vykonávání většiny instrukcí ale způsobuje prodlevu po provedení skoku potřebnou pro znovunaplnění jednotlivých úrovní zpracování. První možností jak čas této prodlevy využít je nepodmíněné provedení pevného počtu instrukcí za instrukcí skoku (delay slots), což vyžaduje speciální podporu ze strany kompilátoru (například MIPS a některé procesory DSP). Druhou možností je včasné dekódování instrukce skoků a započetí načítání instrukčních kódů z cílové adresy skoku. Stále ovšem zůstává problém kterou větev načítat u podmíněných skoků. Řešením je přidání statické (do instrukčního kódu kompilátorem nastavené) nebo dynamické predikce skoků (cache minulých výsledků rozhodování) a spekulativní provádění instrukcí (když dojde k vybrání druhé větve než bylo predikováno jsou změny stavu a registrů zapomenuty nebo navráceny do stavu před rozhodováním).

Druhá cesta ke zrychlení provádění instrukcí je zjednodušení instrukčního souboru, což vede ke snížení počtu fází vykonávání instrukcí a zjednodušení a zrychlení dekódování operací (RISC - Reduced Instruction Set Computer). Protože řešení vzájemných závislostí mezi daty po sobě následujících instrukcí způsobuje další potíže při postupném a paralelním výkonu instrukcí, využívá se takzvaného přejmenovávání registrů, kdy je uchovávána minulá hodnota registrů pro dokončení paralelně vykonávaných předchozích instrukcí. Zároveň se s výhodou používá tato technika pro ukládání výsledků spekulativního provádění instrukcí. Pro další snížení počtu závislostí je v některých procesorech eliminován příznakový registr (například Alpha firmy Digital). Většina nových procesorů též omezuje možnost provádění aritmetických a logických operací pouze na operace mezi registry, což vede k dalšímu snížení závislostí instrukcí. Tyto architektury mají pouze omezený počet instrukcí pro přístup k vnější paměti a proto se nazývají architekturou load-store. Tato redukce komplexnosti instrukcí a použití pevné délky instrukčního kódu vede k nižší hustotě kódu a k nárůstu délky programů. Tím se kruh problémů s vývojem procesorů v podstatě uzavírá. Vznikají

procesory, které mohou často používané 32-bitové kódy instrukcí vyjádřit též 16-bitovými zkrácenými kódy (ARM) nebo zpracovávají kódy proměnné délky (16, 32 nebo 48 bitů u procesorů ColdFire). Další možností je využití velmi dlouhých kódů instrukcí s poli pro několik operací a informacemi o vzájemných závislostech jednotlivých operací (například IA-64 Itanium a některé procesory DSP). Problém nízké hustoty strojového kódu procesorů RISC lze dokumentovat na vývoji další generace procesorů Motorola pro řídicí aplikce. Pro velmi náročné řídicí aplikace jsou mikrokontroléry ColdFire/68000 nahrazovány mikrokontroléry na bázi PowerPC. Architektura PowerPC vychází ze sálových počítačů firmy IBM s procesory Power, u kterých nebylo potřeba na paměti pro kód programů šetřit. Protože integrované mikrokontroléry (MPC566) nemají paměti FLASH nazbyt, přistoupila firma Motorola ke statické Huffmanově dekompresi kódu při načítání instrukcí jednotkou řízení sběrnice.

Z výše uvedeného výkladu vyplývá, že neexistuje ideální architektura mikroprocesorů a že pro každou oblast využití mikroprocesorů je nutné posoudit všechny klady a zápory jednotlivých architektur. Složitost výběru mikroprocesorů a mikrokontrolérů pro řídící aplikace je o to větší, že řešení musí být většinou relativně levná, potřebovat co nejméně paměti, mít malé energetické nároky a mít vhodné vlastnosti pro snadnou implementaci řízení v reálném čase a zpracování různých signálů a dat.

3 Mikroprocesory Motorola 680x0

3.1 Vznik a vývoj architektury 680x0

Architektura rodiny procesorů Motorola 680x0 byla navrhována s velkým výhledem do budoucnosti. Již první mikroprocesor 68000 s pracovní frekvencí 8MHz obsahoval vnitřně zcela 32 bitovou architekturu. Protože největší dosažitelné pouzdro v době vzniku mikroprocesoru (rok 1979) bylo pouzdro DIL64 a třicetidvoubitová šířka sběrnic a pamětí byla těžko realizovatelná, byla vnější datová sběrnice pouze 16-ti bitová a vyvedeno bylo pouze 24 bitů adresové sběrnice. Později byla uvedena redukovaná verze 68008 s 8 bitovou datovou sběrnicí a 20 bitovou adresovou sběrnicí a vnitřně rozšířená verze 68010. Verze 68HC001 a 68EC000 umožňovaly při resetu nastavení do osmibitového nebo šestnáctibitového režimu sběrnice. Prvním mikroprocesorem s třicetidvoubitovou datovou i adresovou sběrnicí byl až mikroprocesor 68020.

Při adresaci nejsou používány žádné segmentové registry. Třicetidvoubitová adresa operandů je tvořena prostým součtem registrů a případných ofsetů. Velikost datových struktur je limitována pouze velikostí celého adresového prostoru. Procesory ukládají nejvýznamější byte delších operandů do paměťové buňky s nejnižší adresou (Big-Endian).

Mikroprocesory 680x0 obsahují 16 třicetidvoubitových registrů rozdělených na 8 registrů datových (D0 až D7) a 8 registrů adresových (A0 až A7). Datové registry mohou obsahovat 32, 16 a 8 bitové operandy všech operací a dále 32 nebo 16 bitový zdroje ofsetu od adresy uložené v adresovém registru. Operace s adresovými registry jsou omezené na přesuny, sčítání, odčítání, komparace a tvorbu báze i indexu adresy operandů. Adresové

registry umožňují pouze 32 bitové a 16 bitové operace a tvoří-li cíl 16 bitové operace je hodnota automaticky znaménkově rozšířena na 32 bitů. Jediný z těchto registrů, který má speciální určení, je registr A7, který slouží jako ukazatel na vrchol zásobníku návratových adres využívaný též k dočasnému ukládání hodnot registrů a stavu procesoru při obsluze přerušení a výjimek. Zásobník roste (je naplňován) směrem k nižším adresám, což je výhodné pro předávání parametrů při volání podprogramů a alokaci lokálních proměnných.

Mikroprocesor rozlišuje uživatelský (user) a systémový (supervisor) režim a pro každý režim je vyhrazen vlastní ukazatel zásobníku. K aktuálnímu ukazateli zásobníku lze vždy přistupovat jako k registru A7 (též označován jako SP). V systémovém režimu A7 odpovídá systémovému ukazateli zásobníku SSP (supervisor stack pointer) a uživatelský ukazatel zásobníku je přístupný přes registr USP (user stack pointer). Některé instrukce mohou být prováděny pouze v systémovém režimu. Zároveň informace o druhu přístupu a režimu procesoru je předávána spolu s adresou při přístupu do vnější paměti a může být využita externí jednotkou správy paměti k ochraně nebo separaci systémového adresového prostoru. Jediný možný přechod z uživatelského do systémového režimu je při obsluze výjimek (softwarově vyvolané k tomu určenou instrukcí, vnější přerušení a způsobené chybami při provádění instrukcí - ochrana paměti, ilegální instrukce atd.). Každé výjimce je přiřazen jeden z 256 třicetidvoubitových vektorů uložených v paměti od adresy nastavené v registru VBR (po startu procesoru obsahuje 0). Procesory nižší než 68010 nemají registr VBR (vector base register) implementovaný a tabulka vektorů pro ně začíná vždy na adrese 0. Přes tabulku vektorů je vyřešena i inicializace procesoru po deaktivaci signálu RESET. Do registru SSP je přenesena hodnota uložená ve vektoru 0 (adresa 00h) a čítač instrukcí je nastaven na hodnotu vektoru 1 (adresa 04h) ukazujícího na na první instrukci, od které bude v systémovém režimu po inicializaci procesor startovat. Vnější přerušení mohou mít prioritu 1 až 7 (priorita 0 složí k zakázání přerušení) a mohou v potvrzovacím cyklu předat hodnotu vektoru přerušení nebo využít automatického přiřazení vektoru přerušení podle priority nastavením signálu AVEC.

Procesor 68010 (1982) přidal drobné ale podstatné vylepšení, na rozdíl od svých předchůdců byl schopen dokončit nebo zopakovat operaci, která byla přerušena výjimkou (především chyba adresy nebo sběrnice). Po přidání vnější jednotku správy paměti (68451) pak bylo možné implementovat virtuální paměť. Dále byl přidán cyklický režim automaticky použitý při provádění cyklu obsahujícího jednu jednoslovnou instrukci bez nutnosti opakovaného načítání instrukčního kódu.

Procesor 68020 (1984) přidal třicetidvoubitové sběrnice, třífázové postupné zpracování instrukcí a 256 bytů velkou instrukční vyrovnávací paměť cache. Zároveň byly v mikrokódu implementovány instrukce pro přímé připojení i více koprocesorů. Nejdůležitějšími byly matematický koprocesor 68881/68882 a obvod 68851 pro správu virtuální paměti. Dále bylo přidáno dynamické řízení šířky sběrnice pro 32, 16 a 8 bitové paměti a periferie, podpora nezarovnaného přístupu na data (například 16 bitů na liché adrese) a další adresní režimy. Procesor 68030 (1987) měl již jednotku správy paměti integrovánu přímo na čipu a k 256 byte instrukční paměti cache přibylo 256 byte datové cache. Procesor 68040 (1991) již obsahoval 4 kB datové a 4 kB instrukční chache s jednotkou správy paměti, šestifázové postupné zpracování instrukcí a matematický koprocesor na čipu. Vnější sběrnice podporovala

na rozdíl od 68020 a 68030 pouze synchronní třicetidvoubitový režim s udržováním konzistence obsahu datové paměti cache s dalšími iniciátory přenosů na sběrnici. Pro vyvedení sběrnice s dynamickou šířkou pro periferie lze přidat obvod 68150.

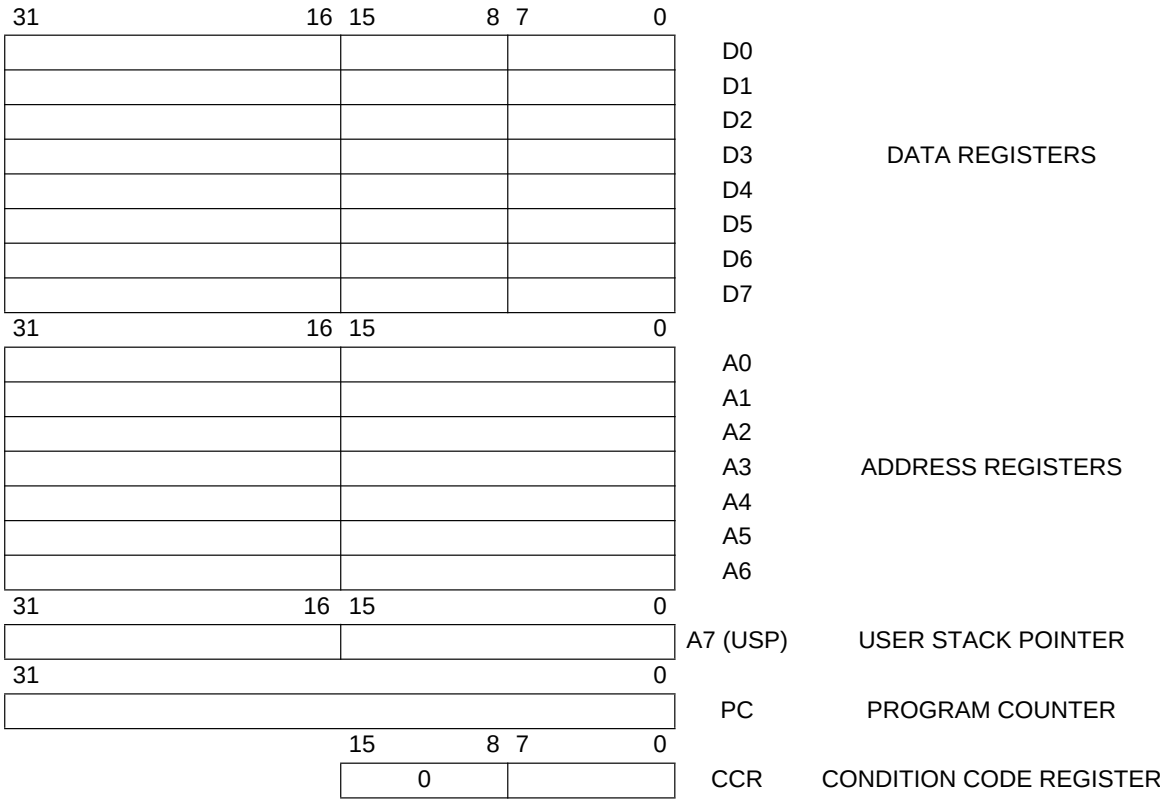
Posledním členem klasické řady 680x0 je mikroprocesor 68060 (1994). Jedná se o superskalární mikroprocesor, který je schopen vykonávat až dvě celočíselné operace nebo jednu celočíselnou operaci a jednu operaci v plovoucí řádové čárce v jednom hodinovém taktu. Ve třetí fázi desetifázového zpracování instrukcí jsou instrukce dekódovány do podoby odpovídající procesorům RISC a ukládány do 16 položkové fronty ve čtvrté fázi zpracování. V těchto fázích zpracování jsou s použitím paměti historie větvení a skoků (branch cache) řešeny změny instrukčního toku. Instrukce jsou dále předávány do dvou řetězců zpracování s fázemi dekódování, generování adresy a načtení operandů. Nakonec jsou předány do dvou celočíselných výkonných jednotek (nebo jedné celočíselné a jedné pro operace v plovoucí řadové čárce) a výsledky jsou předány do dvou zápisových jednotek (writeback). Instrukční i datová cache jsou zdvojeny na 8 kB.

Procesor 68060 obsahoval množství dalších novinek. Hodiny pro jednotlivé výkonné jednotky byly dynamicky odpojovány při cyklech, kdy jednotka v daném cyklu nebyla použita, procesor pracoval s napájením 3.3 voltu, návrh umožňoval snížení i zastavení hodin a pro snížení výkonu bylo možné jednu výkonnou jednotku zastavit. Spotřeba procesoru činila 4-6 watů. Dalšího zrychlení výkonu instrukcí bylo dosaženo tím, že jednoduché operace mezi registry bez přístupu do paměti byly vyhodnoceny a dokončeny o dvě fáze zpracování dříve již ve fázi výpočtu adresy.

Protože se procesory z řady 680x0 osvědčily i v řídicích aplikacích, bylo vyvinuto množství mikrokontrolérů přímo určených pro průmyslové aplikace, automobilový průmysl, komunikací zařízení a spotřební elektroniku. Původně z integrace mikroprocesorového jádra 68EC000 s různými periferiemi vznikla rodina 683xx určená pro řídicí aplikace. Motorola přímo velkým odběratelům nabízela knihovnu subsystémů, ze kterých bylo možné vytvořit aplikačně specifický mikrokontrolér. Množství z těchto kombinací bylo obsaženo i v standardní nabídce. V dnešní době ovšem cena za naintegrování více periferií na jeden čip poklesla a tak množství kombinací bylo nahrazeno vždy jedním mikrokontrolérem obsahujícím většinu periferií určených pro danou oblast použití.

Dlouhodobou stálicí v oblasti řídicích systémů se stal mikrokontrolér 68332, který je již založen na modernějším mikroprocesorovém jádře CPU32. Toto přejímá některé z vlastností mikroprocesoru 68020. Především precizní obsloužení výjimek, cyklický režim, plně 32 bitové násobení a dělení, 32 bitové bázové offsety a škálování i potlačení indexu při adresaci. Tento mikrokontrolér je podrobněji popsán v samostatné kapitole. Přidáním rozhraní CAN, AD převodníku a dalších čítačů vznikl mikrokontrolér 68376, který je již asi posledním a nejúplnějším členem s architekturou 683xx určeným pro oblast řídicích aplikací.

Oblast komunikací je v rodině 683xx zastoupena především mikrokontrolérem 68360, který obsahuje pomocný mikrořadič RISC s DMA kontrolérem obsluhujícím čtyři konfigurovatelná sériová rozhraní schopná implementovat množství průmyslových standardů (Profibus, BITBUS atd.). Verze 68360EN je připravena i pro komunikaci v sítích ETHERNET. Použité mikroprocesorové jádro CPU32+ má vyvedenou již plně 32 bitovou datovou sběrnici a činí tak mikrokontrolér 68360 nejvýkonnějším členem rodiny 683xx.



Obrázek 1: Uživatelský model CPU32

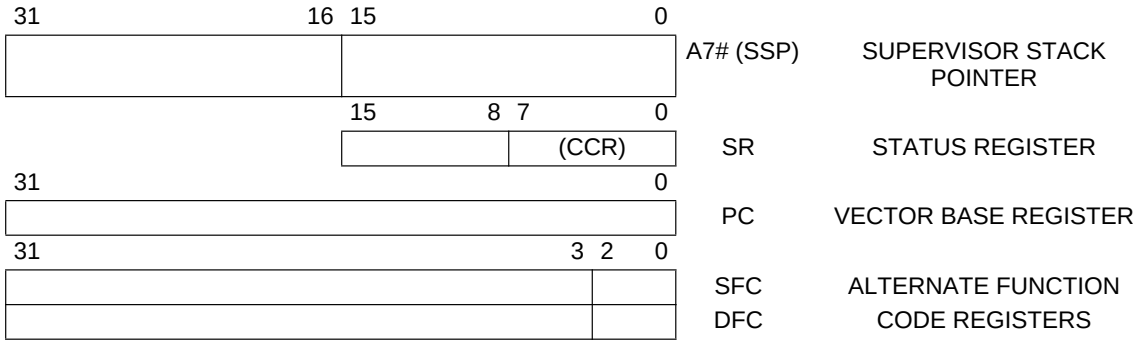
Mikrokontroléry s jádrem EC000 byly použity pro svůj nízký příkon v množství kapesních kalkulátorů a počítačů. Například nahradily procesory Z80 v kalkulátorech firmy Texas Instruments. Integrovaná verze DragonBall 68VZ328 se již několik let používá v organizérech Palm.

3.2 Programový model procesorů 68xxx

Následující odstavce popisují základní charakteristiky architektury 68xxx z programátorského pohledu. Pro úplný popis je nutné prostudovat manuály určené pro konkrétní mikroprocesor nebo mikrokontrolér. Zde předkládaný rozsah informací by měl posloužit jako úvod ke snadnější orientaci v těchto manuálech nebo pro samostatnou práci v rámci již připraveného softwarového prostředí s hlavním zaměřením na použití mikrokontrolérů s jádrem CPU32 pro řídicí aplikace. To je také oblast, ve které má architektura 68xxx stále podstatný význam.

3.2.1 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



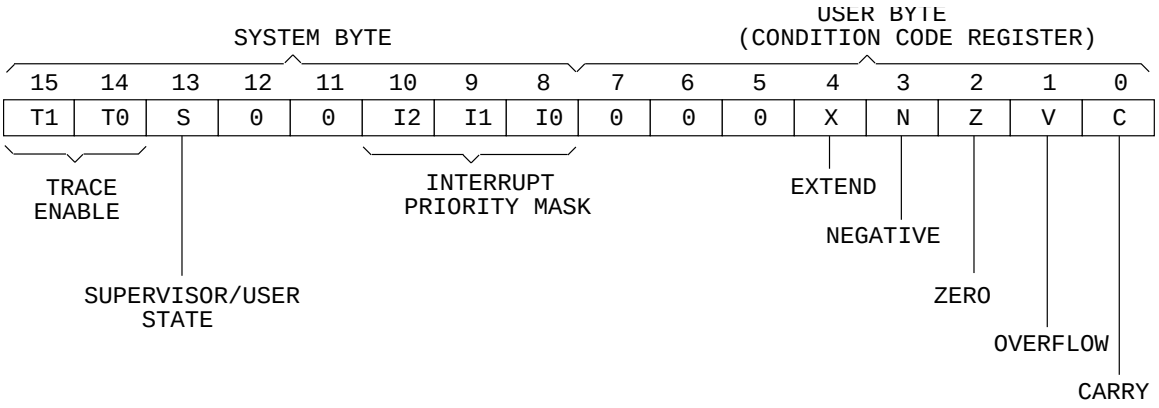
Obrázek 2: Systémový model CPU32

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 do paměti pouze v kombinaci s adresovým registrem jako index (hodnotu adresového registru lze ovšem u CPU32, 68020 a vyšších potlačit). Adresové registry lze použít jak jako index tak 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 výjimek (softwarově vyvolané k tomu určenou instrukcí, vnější přerušení a způsobené chybami při provádění instrukcí - ochrana paměti, ilegální instrukce atd.). Každé výjimce je přiřazen jeden z 256 třicetidvoubitových 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. Vprocesorech 68020 byla přidána ještě jedna kopie registru A7 a další procesorový režim pro obsluhu hardwarových přerušení. Jeho užitečnost se však neprokázala a v procesoru 68030 již implementován nebyl.

Stavový registr (SR) procesoru 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 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 výjimky po provedení jedné instrukce. CPU32 umožňuje i trasování, při kterém dojde k výjimce pouze při změně toku vykonávaných instrukcí (například skok, návrat, volání). Maska přerušení povoluje procesoru přijmout pouze vnější požadavky na přerušení s vyšší než maskovanou prioritou. Výjimku představuje pouze vnější úroveň 7, která je použita pro nemaskovatelná přerušení.

Bity podmínkové části (CCR) stavového registru jsou nastavovány podle výsledků logickými, aritmetickými a bitovými operacemi a operacemi rotací, posunů a přesunů. Příznak X slouží k rozšiřování operací na operandy větší délky než 32 bitů. Příznaky N a Z informují



Obrázek 3: Stavový registr

o záporném nebo nulovém výsledku předchozí operace. Indikace přetečení a přenosu jsou uloženy do příznaků V a C. Příznaky nejsou ovlivněny operacemi, které ukládají výsledek do adresových registrů.

Většina kompilátorů využívá registr A6 jako ukazatel na rámec zásobníku aktuální funkce. Registr A5 bývá použit pro implementaci kódu nezávislého na svém umístění - PIC (Possition Independent Code). Registr D0 a pro 64 bitové výsledky i D1 slouží pro předání návratové hodnoty funkcí.

3.2.2 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

3.2.3 Režimy adresace

Následuje krátký výčet jednotlivých režimů adresace mikroprocesorů 68xxx a 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 (případně jen jeho nižších 16 bitů), pro procesory CPU32 a 68020+ může být index násoben číslem 1, 2, 4 nebo 8

(bd,An,Xn*SCALE) adresa je vytvořena ze součtu adresového registru s indexovým registrem násobeným měřítkem **SCALE** (1, 2, 4 nebo 8) a bázovým posunutím délky až 32 bitů (0, 16 nebo 32), kódování režimu umožňuje potlačit hodnotu indexu, případně i adresového registru, tento režim adresace je implementován v procesorech CPU32 a 68020+

(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, další možnosti jsou shodné s režimy vztaženými k adresovým registrům

Další rozšířené režimy adresace jsou implementovány pouze v procesorech 68020 až 68060

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

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

3.2.4 Instrukční soubor

Základní instrukční soubor procesorů 68xxx obsahuje přibližně 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. Procesory obsahující nebo doplněné o koprocesor pro operace v plovoucí řádové čárce mohou pracovat i s 32 a 64 reprezentovanými reálnými čísly a jejich instrukční soubor je patřičně rozšířen. Základní instrukční soubor obsahuje obvyklé aritmetické, logické, bitové, přesunové a řídící 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.

Kódování instrukcí rodiny procesorů 68000 je velmi kompaktní a bylo již při svém vzniku v roce 1979 připraveno pro postupné rozšiřování. Instrukce se skládají z jednoho nebo více 16 bitových slov. Základní informace o požadované operaci i informace o použitých registrech a základních adresových režimech jsou obsaženy již v prvních 16-bitovém slovu. Pouze několik operací jako je 32-bitové násobení a dělení využívá k doplnění informací o třetím použitém registru další slovo s doplňujícími informacemi. Všechny ostatní instrukce potřebují více slov pouze pokud vyžadují přímé konstantní hodnoty (32, 16 nebo 8 bitů, přičemž 8 bitů je uloženo v celém slovu), absolutní adresy (16 nebo 32 bitů) nebo využívají indexové adresace, která vyžaduje minimálně jedno slovo s rozšířením. Adresní režim a použitý registr jsou uloženy v 6-bitovém poli v instrukčním kódu označovaném jako efektivní adresa **EA** reprezentující libovolný ze 14 adresních režimů (znázorněno v obrázku 5). Pole pro uložení efektivní adresy obsahuje tři bity pro uložení módu adresace (MODE) a další tři bity informují o použitém bázovém adresovém registru (REGISTER). Absolutní režimy a relativní adresování vzhledem k PC tuto informaci nepotřebují, a

SINGLE EA INSTRUCTION FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS					
										MODE			REGISTER		

BRIEF FORMAT EXTENSION WORD

15	14	12	11	10	9	8	7	0
D/A	REGISTER		W/ L	SCALE		0	DISPLACEMENT	

FULL FORMAT EXTENSION WORD(S)

15	14	12	11	10	9	8	7	6	5	4	3	2	0
D/A	REGISTER		W/ L	SCALE		1	BS	IS	BD SIZE		0	I/IS	
BASE DISPLACEMENT (0, 1, OR 2 WORDS)													

Field	**Definition**	**Field**	**Definition**
Instruction Register Extension Register	General Register Number Index Register Number	BS	Base Register Suppress 0 = Base Register Added 1 = Base Register Suppressed
D/A	Index Register Type 0 = Dn 1 = An	IS	Index Suppress 0 = Evaluate and Add Index Operand 1 = Suppress Index Operand
W/L	Word/Long Word Index Size 0 = Sign−Extended Word 1 = Long Word	BD SIZE	Base Displacement Size 00 = Reserved 01 = Null Displacement 10 = Word Displacement 11 = Long−Word Displacement
Scale	Scale Factor 00 = 1 01 = 2 10 = 4 11 = 8	I/IS *	Index/Indirect Selection Indirect and Indexing Operand Determined in Conjunction with Bit 6, Index Suppress

*Memory indirect addressing will cause illegal instruction trap; must be = 000 if IS = 1

Obrázek 4: Kódování instrukce CPU32 s jednou efektivní adresou

proto jsou kódovány shodnou hodnotou na pozici MODE a k vzájemnému rozlišení slouží rozdílná hodnota na pozici REGISTER. Pro indexové adresování je nutné přidat informaci v rozšiřujícím slově o indexovém registru (Xn může být libovolný Aa nebo Dn - pole D/A a REGISTER), osmibitovém znaménkově rozšířeném ofsetu (DISPLACEMENT), počtu významných bitů indexu (16/32 podle W/L) a násobiteli indexu 1, 2, 4 nebo 8 (pozice SCALE). Procesory 68020 a výše povolují i FULL EXTENSION formát, který umožňuje zakódovat i ofsety 16 a 32 bitů, potlačovat bázový, indexový registr i ofset a zvolit dvojité nepřímé adresování.

Většina aritmetických a logických dvouoperandových instrukcí obsahuje druhé tříbitové pole s informací o registru představujícím druhý operand a o směru provádění operace (registr+**EA** → registr nebo **EA**+registr → **EA**). Velmi výkonná instrukce **MOVE** jako jediná obsahuje dvě pole **EA** pro zdroj a cíl a informaci o délce přenášených dat (8, 16 nebo 32 bitů). Instrukce pro naplnění cíle znaménkovou 8-bitovou konstantou a přičtení nebo odečtení čísla 1 až 8 je též možné zakódovat v rychlém tvaru do jednoho 16-bitového slova.

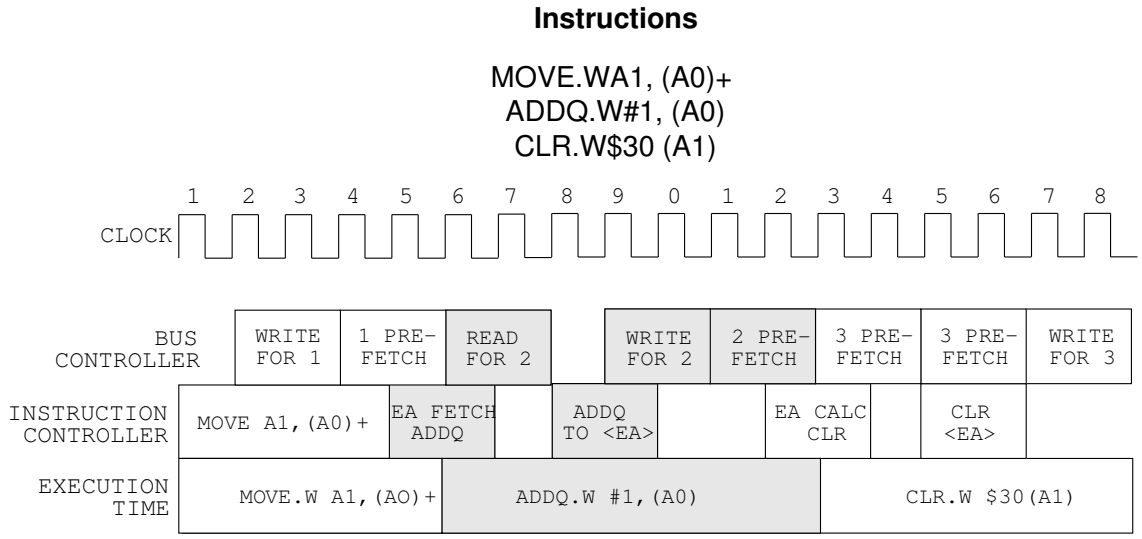
Pro zrychlení blokových přesunů a práce s řetězci obsahuje procesor instrukci pro tvorbu podmíněných a nepodmíněných cyklů s počtem cyklů v 16-bitech datového registru. Při kombinaci této instrukce s libovolnou instrukcí s délkou instrukčního kódu 16-bitů přejde CPU32 do smyčkového módu, kdy není výkon smyčky zatěžován opakovaným načítáním instrukčních kódů z paměti.

Výkon instrukcí v procesorech CPU32 je rozdělen do několika fází. Jednotka řízení sběrnice se v době vykonávání minulé instrukce postará o načtení instrukčního kódu, který je poté předdekódován výkonnou jednotkou, jednotka sběrnice se postará o načtení případných dalších slov instrukčního kódu a zdrojových operandů. Poté dojde ve výkonné jednotce k provedení instrukce a data k zápisu, je-li to potřeba, jsou předána jednotce řízení sběrnice, která provede zápis. Ten se již může překrývat s výkonem další instrukce. Příklad průběhu výkonu instrukcí je znázorněn na obrázku 5. Příklad předpokládá nejrychlejší dvoutaktový přístup k paměti dat i programu.

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

3.3 Sběrnice mikroprocesorů 68000 až 68030

Sběrnice mikroprocesorů se postupně vyvíjela. Rozhraní mikroprocesoru 68000 tvoří 16-ti bitová asynchronní sběrnice. Procesor potvrzuje signálem AS platnost adresy A0 až A23, informace o směru přenosu (signál R/W) a druhu přístupu (signály FCx rozlišují přístupy user/supervisor, data/code, potvrzení přerušení a procesorové přenosy). Signály UDS a LDS určují zda se v daném cyklu využívá horní (D8-D16) a nebo dolní (D0-D7) část datové sběrnice. Zároveň UDS a LDS určuje platnost zapisovaných dat. Periferie nebo subsystém pamětí musí potvrzovat úspěšný přenos signálem DTACK, neúspěšný přenos je nutné ukončit signálem BERR. Přidané signály E, VMA a VPA umožňovali k procesoru připojit i starší periferie z osmibitové stavebnice 6800. U verze 68EC000 byl přidán signál MODE, kterým je možné po spuštění procesoru zvolit osmibitový režim sběrnice. Signály BR (bus request - požadavek), BG (bus grant - předána) a BGACK (potvrzení přijmutí)

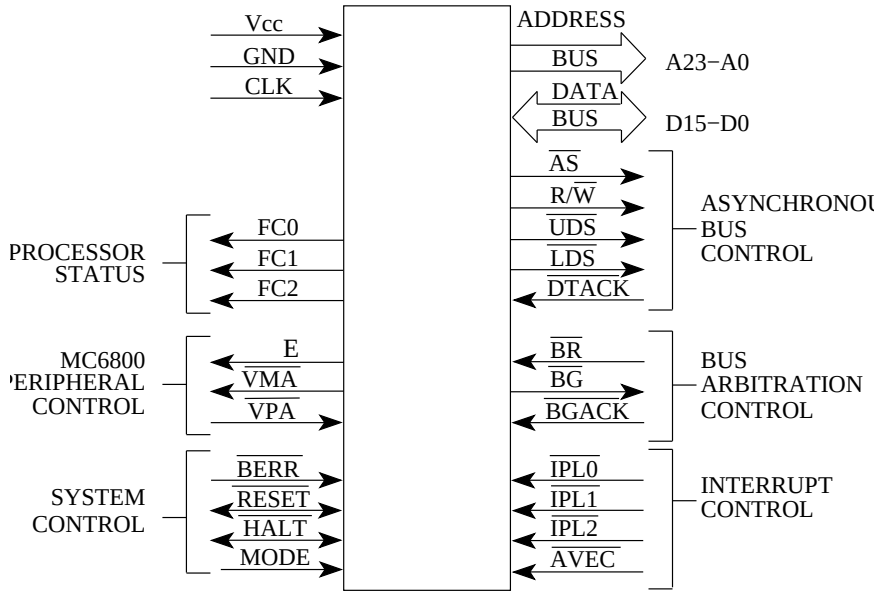


Obrázek 5: Průběh vykonávání instrukcí CPU32

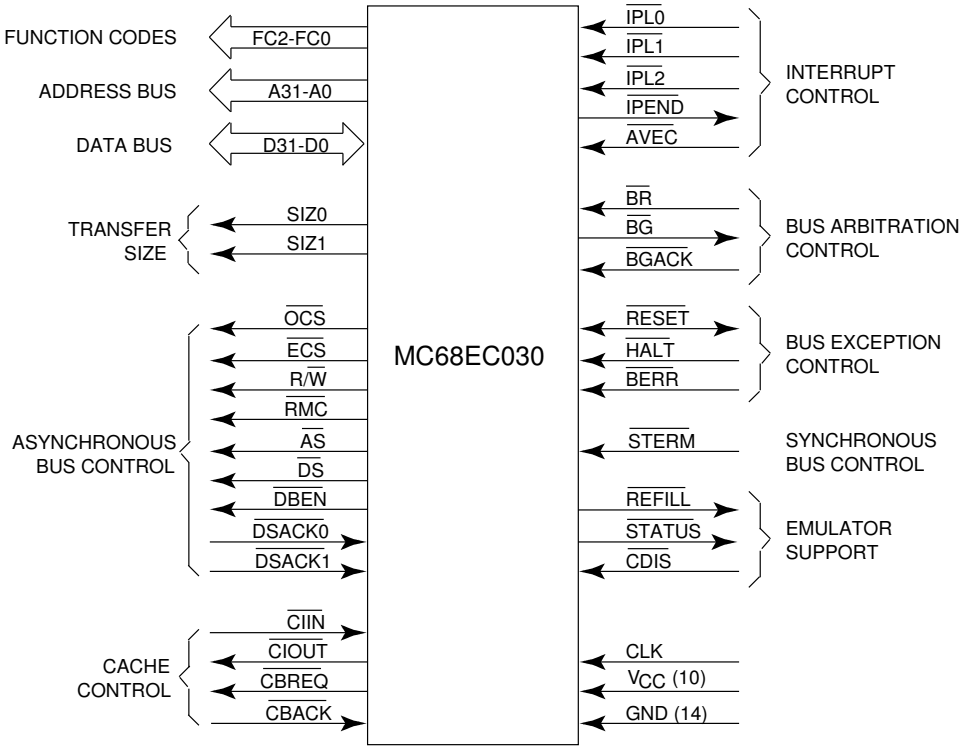
slouží pro koordinaci přístupů více procesorů ke společné sběrnici. Poslední skupinou jsou signály pro žádost o vnější přerušení úrovně 1 až 7. Pokud je priorita požadavku (IPLx) vyšší než hodnota uložená v masce přerušení, přejde procesor do cyklu potvrzení přerušení a započne cyklus čtení vektoru přerušení z žádající periferie. Periferie může vyslat a potvrdit číslo vektoru/výjimky přes datovou sběrnici a signál DSACK nebo může aktivovat signál AVEC a vyjímka je přiřazena automaticky podle priority požadavku. Sběrnice procesorů 68010 je podobná.

Procesory 68020 a 68030 uvedli sběrnici s jednotkou pro nezarovnané přístupy a dynamickým testováním šířky sběrnice připojené periferie až během přístupového cyklu. Signály SIZ0 a SIZ1 informují o požadované délce přístupu v bytech. Periferie informuje procesor o skutečné šířce provedeného přenosu přes potvrzovací signály DSACK0 a DSACK1. Pokud je to potřeba opakuje přenos zbývajících bytů automaticky kontrolér sběrnice v mikroprocesoru s patřičně zvýšenou adresou. Protože procesory pracují v režimu big-endian je nutné periferie s užší šířkou sběrnice zapojovat na vyšší část datové sběrnice (šestnáctibitové periferie k vodičům D16 až D31, osmibitové k D24 až D31). Při zápisu procesor nemá dostupnou dopředu informaci, na kterých vodičích bude přístup na adresy s nenulovým A0 nebo A1 probíhat, proto zapisovaná data zrcadlí takovým způsobem, aby již v prvním cyklu proběhla nejdelší možná část přenosu. Asynchronní přístupy trvají minimálně tři hodinové cykly.

Procesor 68030 přidává i další režim pro rychlý dvoucyklový synchronní přístup do paměti (potvrzený signálem STERM), řízení oblastí se zakázaným použitím vyrovnávacích pamětí (CIIN a CIOUT) a možnost rychlého blokového plnění řádků interní cache (burst režim, CBREQ a CBACK). V optimálním případě je možné 16 byte dlouho řádku interní cache naplnit v pěti hodinových cyklech. Procesor také přidává i informaci o přijetí požadavku na přerušení (IPEND).



Obrázek 6: Sběrnice mikroprocesorů 68000/EC000



Obrázek 7: Sběrnice mikroprocesoru 68030

3.4 Superskalární procesor 68060

Procesor 68060 přebírá 32-bitovou architekturu i většinu subsystémů vzniklých během vývoje rodiny mikroprocesorů 680x0 a navíc je doplněn o superskalární vykonávání instrukcí. To je také důvod proč proč nebyl uveden podrobnější popis procesorů 68020, 68030 a 68040. Pro dosažení vysokého výkonu bylo nutné i některé subsystémy zjednodušit. Několik složitých instrukcí pro operace s celými čísly a čísly v plovoucí řádové čárce bylo oproti předchozím procesorům vypuštěno a lze je nahradit softwarovou knihovnou. Procesor je vyráběn ve třech variantách:

MC68060 verze obsahující jednotku zprávy paměti (MMU) i koprocesor pro výpočty v plovoucí řádové čárce

MC68LC060 levnější verze bez koprocesoru obsahující jednotku MMU pro aplikace, kde je potřeba ochrana paměti nebo stránkování

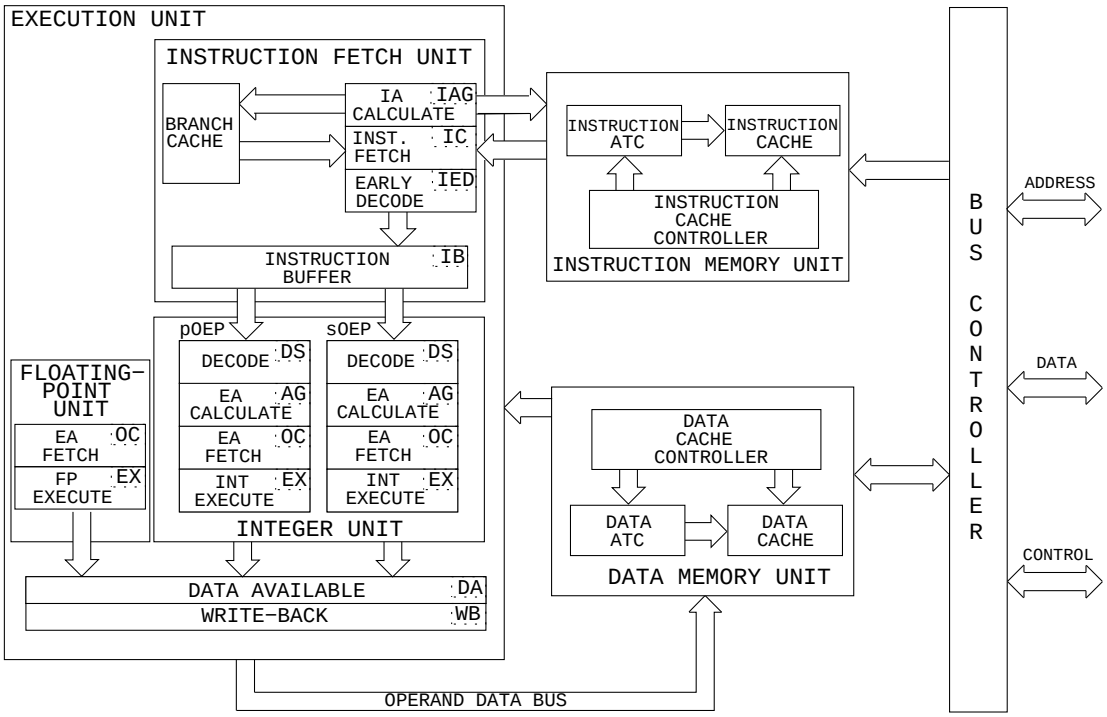
MC68EC060 nejlevnější verze bez koprocesoru i MMU pro embedded aplikace, kde není potřeba ochrana paměti

Procesor je superskalární, vykonává více v programu za sebou jdoucích instrukcí v jednom taktu, a dosahuje výkonu až 100 MIPS při hodinové frekvenci 66 MHz. Jednotka výběru instrukcí načítá instrukce z vnější paměti nebo z instrukční cache paměti. Při predikci adresy pro načtení instrukce je využívána paměť minulých cílů skoků (branch cache) v jednotce výkonu skoků. Instrukce jsou dále zpracovávány ve dvou frontách, což umožňuje vykonat dvě celočíselné operace (nebo jednu celočíselnou operaci a jednu operaci s plovoucí řádovou čárkou) a jednu operaci větvení v každém hodinové periodě.

Paralelní přístup k operandu a kódu instrukce je zajištěn nezávislými pamětmi cache pro data (8 KB) a instrukce (8 KB). Datová paměť cache umožňuje i současný přístup pro čtení a zápis. Změna obsazení paměti cache může být zakázána po dobu časově kritické sekvence kódu pro urychlení výpočtu.

Instrukce proměnné délky jsou předdekódovány s pomocí čtyřfázového zřetězeného zpracování a poté již ve formátu s pevnou délkou ukládány do paměti FIFO. Z této paměti FIFO jsou již instrukce odebírány oběma výkonnými jednotkami, které je ve čtyřech fázích zpracují. Jednotka výkonu skoků zajišťuje, že správně předpovězené skoky nezpůsobí zpoždění o žádný hodinový takt a změna předpokládaného cíle skoku vede k okamžitému znovunaplnění paměti FIFO instrukcí a zpoždění maximálně sedm taktů.

Stránkování virtuální paměti je řešeno podobně jako u procesorů Intel s volitelnou velikostí stránek 8 nebo 4 kB. Každý proces při využití stránkování definuje dva ukazatele na stránkové adresáře. Při aktivaci procesu jsou přeneseny do řídicích registrů URP (user root pointer) a SRP (supervisor root pointer) a slouží k překladu všech virtuálních adres na adresy fyzické. Překlad 32 bitové virtuální adresy je prováděn tříúrovňově (nejvyšších 7 bitů indexuje hlavní stránkový adresář, dalších 7 podřízený stránkový adresář, 5/6 bitů již tvoří index do stránkových tabulek a zbývajících 13/12 bitů reprezentuje offset v rámci stránky). Ukazatele na jednotlivých úrovních mohou informovat o nepřítomnosti



Obrázek 8: Blokové schéma procesoru 68060

cíle a tím umožňují provést patřičnou akci operaccnímu systému v obsloužení výjimky. Odlišností je možnost ve stránkových tabulkách definovat nepřímý odkaz do jiné stránkové tabylky. To umožňuje při sdílení stránky mezi více procesy systému vyhodnocovat na jednom místě příznaky přítomnosti a využití stránky ve všech procesech. Také při odkládání stačí informaci o umístění stránky v externí paměti ukládat jen do jedné stránkové tabulky. Čtyřcestná datová a instrukční paměť cache pracuje již s přeloženou fyzickou adresou. Urychlení překladu adres je docíleno doplněním instrukční a datové překladové (ATC address translation cache) čtyřcestné vyrovnávací paměti (2×64 položek).

Nízká spotřeba procesoru je docílena vnitřním napájením 3.3 V, statickým návrhem logiky a vypínáním hodin pro každou jednotku, která není v daném hodinovém taktu využívána. Procesor je připojitelný přímo k 3.3 voltovým i 5 voltovým pamětem a periferiím.

Vnější sběrnice je nemultiplexovaná, synchronní, plně 32 bitová a pracuje na plné, poloviční nebo čtvrtinové frekvenci hodin procesoru. Signálová logika je kompatibilní s procesory 68040 a umožňuje sledování sběrnice potřebné pro symetrický multiprocessing. Pro urychlení zápisů je řízení sběrnice doplněno o čtyři zápisové vyrovnávací čtyři byte dlouhé registry a jeden 16 byte dlouhý blok pro pozdržení zápisu jedné linky paměti cache při alokaci a čtení nové adresy.

Feature	68000	'EC000	68010	68020	68030	68040	68060
Data bus	16	8/16	16	8/16/32	8/16/32	32	32
Addr bus	23	23	23	32	32	32	32
Misaligned Addr	-	-	-	Yes	Yes	Yes	Yes
Virtual memory	-	-	Yes	Yes	Yes	Yes	Yes
Instruct Cache	-	-	3	256	256	4096	8192
Data Cache	-	-	-	-	256	4096	8192
Memory manager	68451 or 68851			68851	Yes	Yes	Yes
ATC entries	-	-	-	-	22	64/64	64/64
FPU interface	-	-	-	68881 or 68882		Internal FPU	
built-in FPU	-	-	-	-	-	Yes	Yes
Burst Memory	-	-	-	-	Yes	Yes	Yes
Bus Cycle type	asynchronous				both	synchronous	
Data Bus Sizing	-	-	-	Yes	Yes	use 68150	
Power (watts)	1.2	0.13-0.26	0.13	1.75	2.6	4-6	3.9-4.9
at frequency of	8.0	8-16	8	16-25	16-50	25-40	50-66
MIPS/kDhryst.	1.2/2.1	2.5/4.3		6.5/11	14/23	35/60	100/300
Transistors	68k		84k	190k	273k	1,170k	2,500k
Introduction	1979		1982	1984	1987	1991	1994

Tabulka 1: Základní výkonná řada 680x0 pro počítačové systémy

3.5 Srovnání jednotlivých členů rodiny 68xxx

Srovnání výkonosti, šířky a typů sběrnic procesorů z řady 680x0 lze nalézt v tabulce 1. V tabulce 2 je provedeno podobné srovnání s informacemi o integrovaných modulech pro mikrokontroléry 683xx.

4 Mikrokontroléry z řady 68332

Obvod MC68332[1] je 32 bitový vysoce integrovaný mikrokontrolér, který kombinuje jádro s rozsáhlým vstupně výstupním subsystémem. Jednotlivé moduly jsou propojeny vnitřní sběrnicí. 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

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

Feature	68332	68376	68360	68VZ328
Core CPU	CPU32	CPU32	CPU32+	FLX68000
Data Bus	8/16	8/16	8/16/32	8/16
Addr Bus	24	24	32	24/32MB DRAM
Misaligned Addr	-	-	Yes	
Development Int.	BDM	BDM	BDM/JTAG	ICE
TPU (timer)	Yes	Yes		
UART			2xSMC	2x
DRAM controller			Yes	EDO, FP, SD
Static Ram	2K	3.5K+4K	2.5K	
Flash EEPROM				
A/D Converter		8/10 bits		
Serial Ports	1xSCM	1xSCM	4xSCC	
SPI interface	1xQSM	1xQSM	1xSCP	2x
DMA			2 ch	
Timer		CTM4 (8)	4x16, 2x32	2x+2xPWM
Parallel Ports (bits)	up 4 (31)	up 6 (47)	3	10 (78)
Chip Selects X	12	12	8	8
More ...		TouCAN	opt. Ethernet	LCD, RTC
Clock speed Mhz	16/20/25	16/20/25	25/33	up 33
Power voltage	5V	5V	3.3 or 5V	2.7-3.3
Power (watts)	0.6	0.6	0.3-1.0	0.06-0.1
at frequency of	20	20.97	25	33

Tabulka 2: Mikrokontroléry z řady 683xx

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

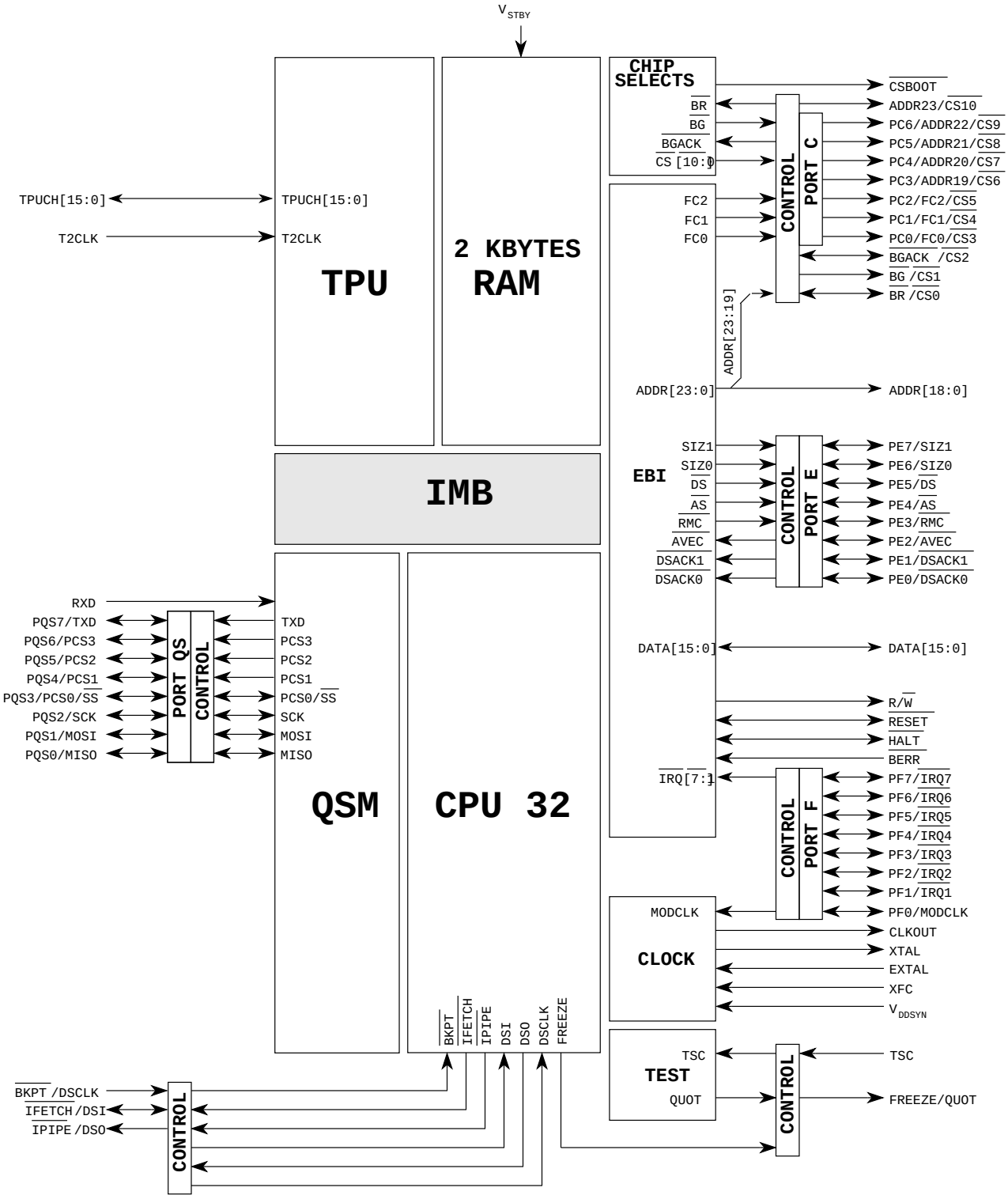
4.1 Přehled vlastností jednotlivých modulů

System Integration Module (SIM) - řízení sběrnic a časování

- 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) - procesorové jádro

- Strojový kód shora kompatibilní s procesory MC68000 a MC68010
- Nové instrukce pro řídící 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 výjimek pro řídící aplikace
- Podporuje trasování do změny toku instrukcí (návrat, volání, ...)



Obrázek 9: Funkční bloky 68332

- 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) - časovací koprocesor

- Obsahuje vlasní 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) - sériová komunikační rozhraní

- Obsahuje sériový interface (SCI), univerzální asynchronní vysílač a přijímač (UART) s volitelným módem, rychlostí a paritou
- Sériový interface (QSPI) 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 - paměť s možností využití pro emulaci TPU **(TPURAM)**

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

4.2 Procesorové jádro 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 zarovnány na sudé adresy. Podrobnější popis programového modelu procesorů 68xxx se zaměřením na CPU32 je popsán v kapitole 3.2.

4.3 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 řídící a systémové funkce.

Interface vnější sběrnice vykonává přenosy mezi vnitřní sběrnicí (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 s rozsahem 2 kB až 1 MB), počáteční adresu, druhy přístupu (čtení, zápis, potvrzení přerušení) a datovou šířku připojené paměti nebo periférie (byte, word, liché byte, sudé byte). Dále se pro každý chipselect volí jestli bude vyžadovat vnější potvrzení přenosu dat nebo jestli bude potvrzení generováno interně po 2 až 16 taktech systémových hodin.

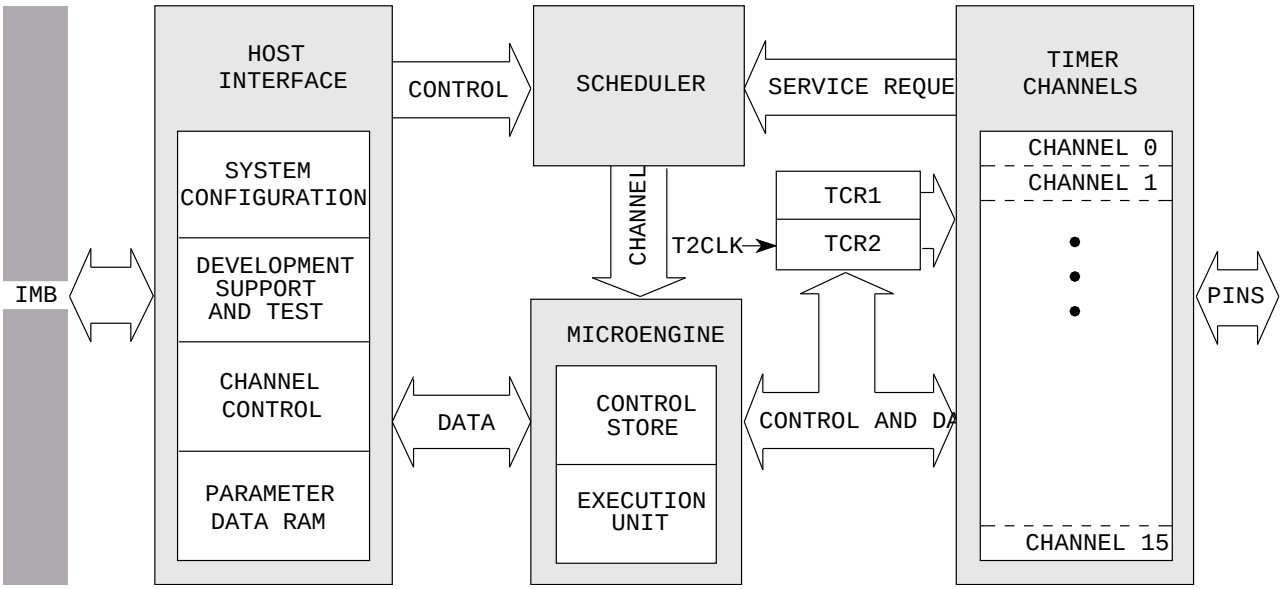
Pokud požadovaná adresa nespadá do oblasti žádného z čipselectů ani neodpovídá žádnému vnitřnímu modulu, je zahájen standardní cyklus sběrnice s vnějším potvrzením akceptace nebo připravenosti dat odpovídající sběrnicovému cyklu procesoru 68020. Potvrzením je možné určit, jestli periferie zpracovala 8 nebo 16 bitů. V případě potřeby (přístup 16 a více bitů, potvrzeno jen 8) řadič dynamické šířky sběrnice dokončí přístup v dalších cyklech. Pokud periferie nebo paměť nepotvrdí data (DSACKx) a ani nenahlásí chybu (BERR), dojde k zablokování systému. Tomu je možné zabránit ochranou selhání sběrnice, která po určitém čase nečinnosti periferie nastaví chybu (BERR) a vyvolá výjimku. Princip nožnosti dynamického určení šířky sběrnice během datového přenosu a architektura big-endian vyžaduje připojení osmibitových periferií na datové piny D8 až D15.

Testovací blok obsahuje logiku pro testování mikrokontroléru při výrobě.

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.

4.4 Časovací koprocesor (TPU)

Časovací koprocesor (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. Dále obsahuje dvě časové základny, 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 komparátorem, komparačním registrem, registrem pro záchyt času změny a 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



Obrázek 10: Časovací koprocesor TPU

kanál je doplněn hardwarem, který dovoluje současné vstupní i výstupní události na všech kanálech

4.5 Komunikační modul (QSM)

Blok synchronní komunikace (QSPI) 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. Vlasní 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.

4.6 Další moduly mikrokontroléru 68336/376

Na obrázku 11 je blokové schéma mikrokontroléru 68376. Tento mikrokontrolér obsahuje všechny moduly mikrokontroléru 68332 (viz) a přidává množství dalších modulů. Tyto

další moduly budou dále stručně popsány.

Standby RAM Module (SRAM) - zálohovatelná paměť

- 4-kilobyty statické paměti RAM

Masked ROM Module (MRM) - prostor pro aplikačně specifický zavaděč

- 8-kilobytů maskou definované paměti ROM přístupné po 8 nebo 16 bitech
- Volitelná počáteční adresa
- Možnost nastavení jako zaváděcí paměti po resetu
- Uživatelem volitelný kontrolní kód

10-Bit Queued Analog-to-Digital Converter (QADC) - analogové vstupy

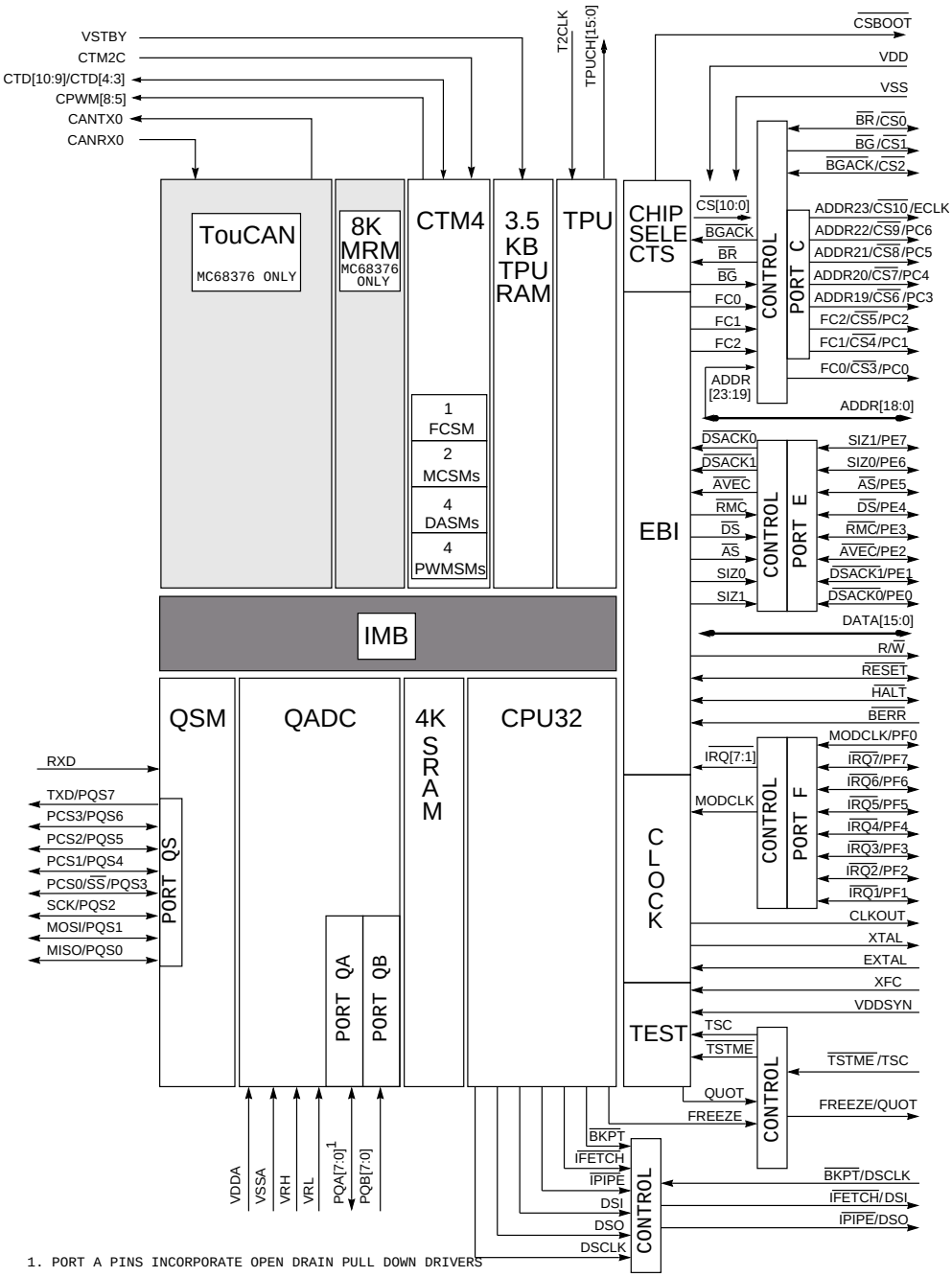
- 16 přímých vstupů; až 44 vstupů s využitím externích multiplexrů
- Šest režimů automatické volby vstupu a režimů převodu
- Paměť pro zadání až 40 požadavků na převody může být rozdělena na dvě cyklické posloupnosti proměnné délky s libovolným množstvím zarážek
- Paměť pro uložení 40 výsledků převodů přístupných ve třech formátech
- Programovatelný čas vzorkování vstupu
- Přímé řízení vnějších multiplexerů

Configurable Timer Module Version 4 (CTM4) - modul čítačů a časovačů

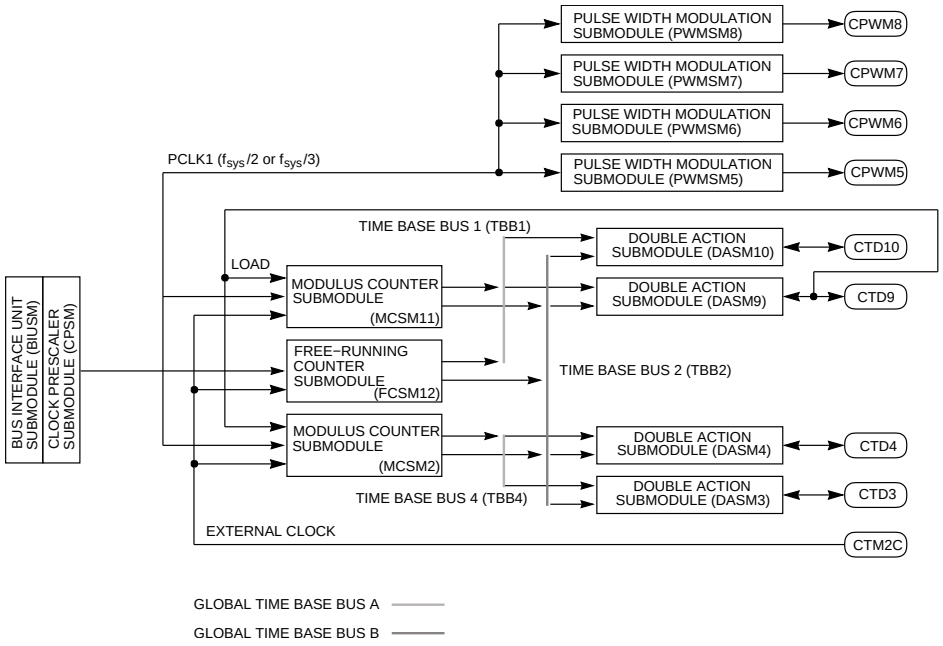
- Dva 16-bitové bloky s předvolbou (MCSMs)
- Jeden 16-bitový volně běžící čítač (FCSM)
- Čtyři moduly se dvěma komparátory nebo záchyty (DASMs)
- Čtyři moduly pro pulzně šířkovou modulaci (PWMSMs)

CAN 2.0B Controller Module (TouCAN)

- Plná implementace specifikace protokolu CAN verze 2.0 A a B
- 16 pozic pro příjem nebo vyslání zprávy s až 8 byty dat
- Společná maska pro filtrování příchozích zpráv pro pozice 0 až 13



Obrázek 11: Funkční bloky 68336/376



Obrázek 12: Konfigurovatelný modul čítačů a časovačů (CTM4)

- Nezávislá masky pro pozice 14 a 15
- Možnost volby pořadí odchozích zpráv na základě nejnižší pozice nebo identifikátoru zprávy
- 16-bitový čítač pro značkování času přenosu zpráv
- Možnost snížení spotřeby s automatickou aktivací od sběrnice

Na rozdíl od mikrokontroléru 68332 byla rozšířena kapacita statické paměti TPURAM na 3,5 kB

4.7 Konfigurovatelný modul čítačů a časovačů (CTM4)

Modul se skládá z několika bloků. Kromě bloku komunikace s mezimodulovou sběrnicí obsahuje modul čtyři bloky generátorů PWM, blok připravující šest různě rychlých hodinových signálů, dále tři bloky generující referenční časové hodnoty a čtyři akční bloky umožňující funkce komparací a záchytu.

Základní hodiny pro všechny bloky mohou být systémové hodiny dělené dvěma nebo třemi. Pro každý blok pulzně šířkové modulace (PWMSM5 až 8) si lze vybrat předdělení základních hodin hodnotou 1,2,...,64 a 256. Dále každý blok obsahuje zdvojené šestnáctibitové registry periody průběhu a délky pulzu, dva komparátory a volbu polarity výstupního signálu.

Další blok připravuje hodinové signály vzniklé dělením frekvence základních hodin hodnotou 1,2,4,8 a 16 a dále jeden signál s volitelným dělitelem 32,64,128 nebo 256. Tyto hodinové signály a jeden vstupní pin mohou sloužit jako zdroj hodin pro trojici bloků, které mohou připravovat tři nezávislé šestnáctibitové časové referenční sběrnice. Dva z těchto bloků (MCSM2 a MCSM11) obsahují registr pro přednastavení délky periody. Načtení hodnoty registru do čítače může být řízeno i libovolnou hranou na jednom ze vstupně výstupních pinů. Poslední blok (FCSM12) pro generování referenční sběrnice je pouze volně běžící šestnáctibitový čítač. Tři časové sběrnice jsou rozděleny na dvě lokální (každá vždy pro dva akční bloky) a jednu globální (přístupná pro všechny čtyři akční bloky).

Každý z akčních bloků (DASM3, DASM4, DASM9, DASM10) obsahuje výběr ze dvou časových sběrnic, čtyři šestnáctibitové registry a logiku řízení akcí prováděných vždy s bloku příslušejícím vstupně výstupním pinem. Tyto bloky lze použít pro následující časové funkce : zastavené čítání, měření délky vstupního pulzu, měření periody vnějšího signálu, zachycení stavu změny vstupu, generování vzestupné a sestupné hrany v definovaných časech, totéž s nastavením příznaku při obou komparacích a nakonec také generování PWM o rozlišení 7, 9, 11, 12, 13, 14, 15 nebo 16 bitů.

Každý z čítačů, záchytů a komparátorů v tomto modulu může volitelně generovat přerušení.

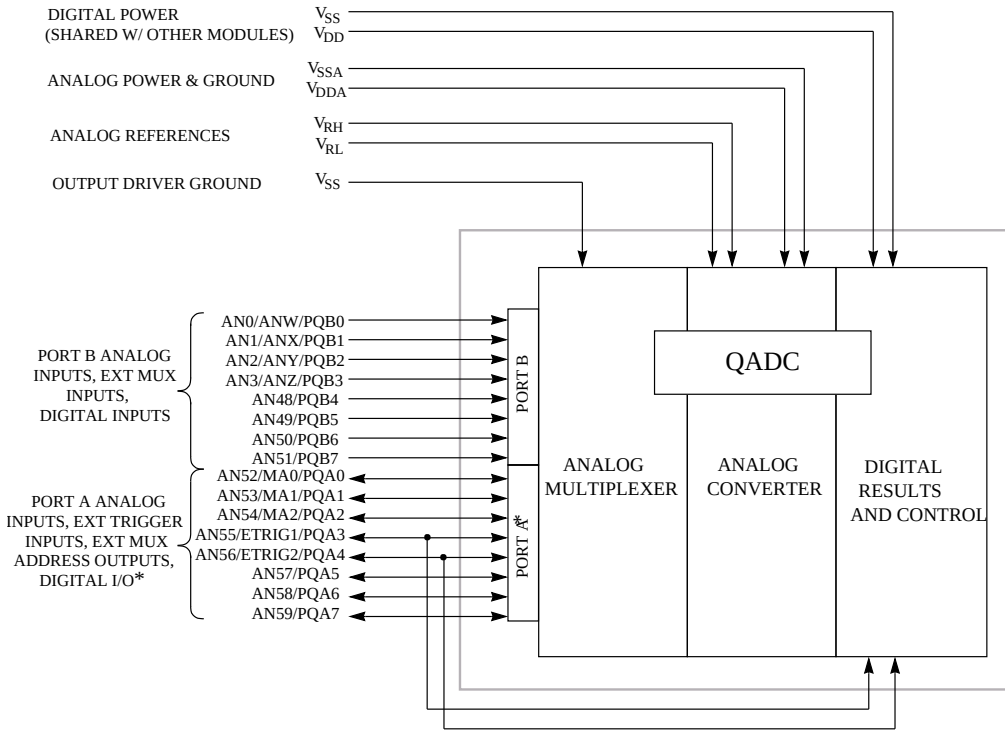
4.8 Inteligentní AD převodník (QADC)

Převodník QADC je schopen automaticky provést až 40 různých převodů a uložit získané výsledky. Jednotlivé převody mohou mít 10 nebo 8-bitové rozlišení, různou délku vzorkování vstupu a další parametry. Přímo k modulu lze připojit 16 analogových vstupů, ale po přidání až osmi externích osmivstupových analogových multiplexerů je modul schopen automaticky adresovat až 41 vstupů. Pořadí převodů je určeno zařazením parametrů a volbou čísla vstupu do jedné ze dvou front převodů. Fronta s vyšší prioritou může přerušit běh fronty s prioritou nižší. Po návratu na nižší prioritu mohou být převody z přerušené fronty dokončeny nebo dojde k restartu převodů od počátku této fronty. Libovolný převod též může mít atribut pro pozastavení běhu fronty a žádosti o softwarovou obsluhu. Fronty požadavků na převody mohou být procházeny cyklicky nebo může být jeden cyklus spuštěn softwarově, z vnějšího pinu nebo od časovače.

Výsledky uložené v paměti RAM přístupné hlavnímu procesoru CPU32 mohou být čteny ve třech formátech, doleva nebo doprava na šestnáct bitů zarovnaný výsledek bez znaménka nebo doleva zarovnaný znaménkový výsledek. Všech šestnáct analogových vstupů může být čteno jako vstupy digitální a osm z nich může též sloužit jako výstupy s otevřeným kolektorem.

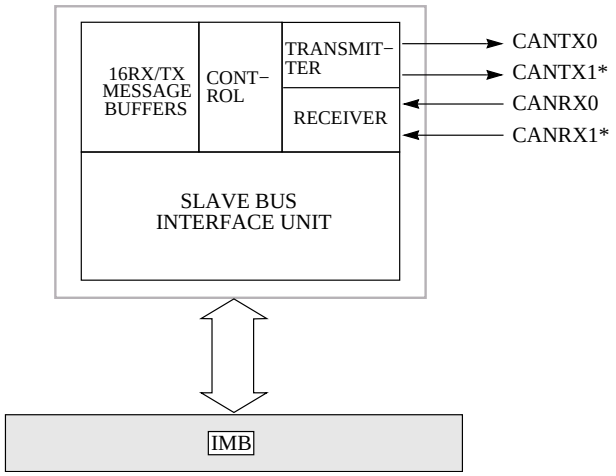
4.9 Modul TouCAN

Dnešní řídící aplikace již nepředstavují pouze jednotlivé řídící systémy, ale většinou využívají více řídících jednotek propojených komunikací. Distribuované řízení snižuje délky výkonových vodičů a v mnoha případech eliminuje nebo zjednodušuje kabeláž senzorů. Pro středně



*PORT A PINS INCORPORATE OPEN DRAIN PULL DOWN DRIVERS.

Obrázek 13: Zapojení vývodů modulu QADC

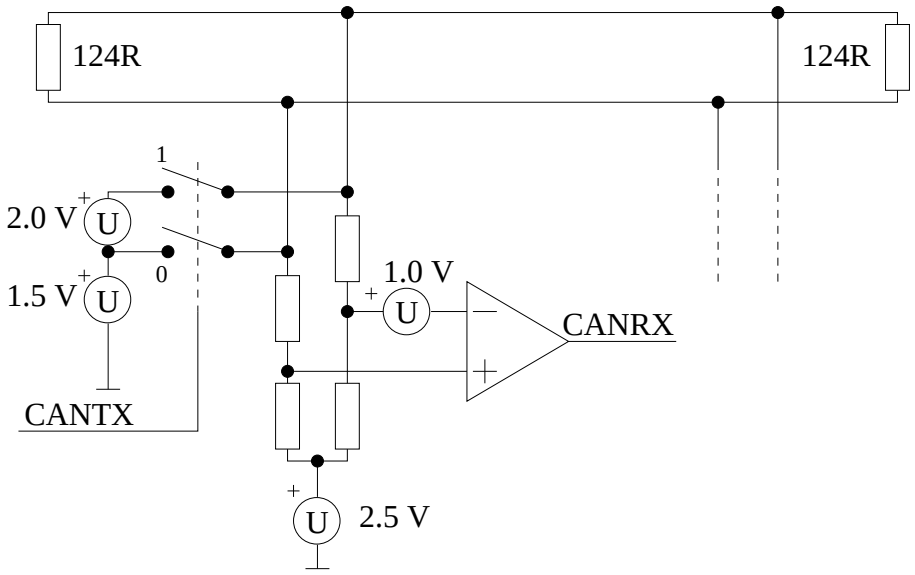


Obrázek 14: TouCAN modul mikrokontroléru 68376

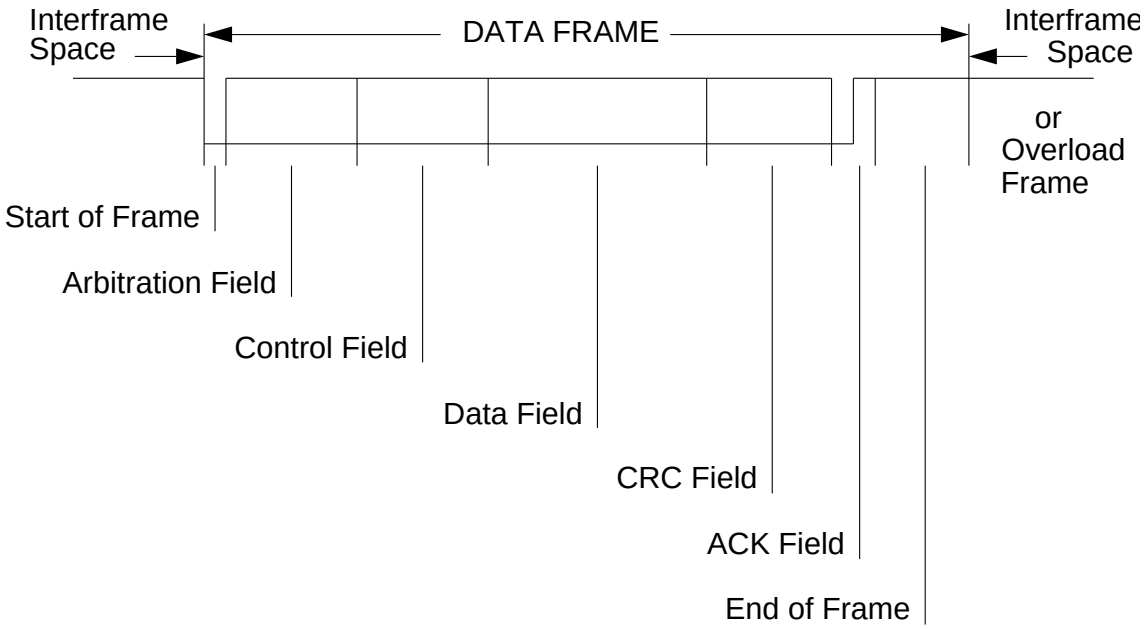
náročné aplikace s velkým množstvím přenášených zpráv při definovaném vztahu rychlosti odezvy se v mnoha oblastech uplatňuje komunikace CAN.

Sběrnice CAN se vyznačuje deterministickou arbitrací přístupu k médiu se staticky definovanými prioritami a způsobem adresace založeným na 11 nebo 29-bitových identifikačních číslech typů zpráv a proměnných (nikoliv přímo dané adresy uzlů). Komunikace probíhá buď v rychlém módu na frekvenci až 1 megabaud nebo v základní konfiguraci do rychlosti 125 kilobaud. Rychlá varianta pracuje se sběrnicovou architekturou do délky krouceného dvoupáru okolo 40 metrů. Médium je na obou koncích zakončeno odporem 124 Ω. Logické hodnoty jedna a nula jsou přenášeny recesivním a dominantním stavem na sběrnici. Vysílaný recesivní stav odpovídající odpory udržovanému napětí 2.5 Voltů na obou vodičích je potlačen i jediným vysílačem vysílajícím dominantní odpovídající rozdílu napětí mezi vodiči většímu než 1 Volt. Vysílání dominantního stavu je docíleno připojením jednoho z vodičů k napětí 1.5 Voltu a druhého k napětí 3.5 Voltu, jak je znázorněno na obrázku 15. Logická funkce wired-and slouží k jednoznačnému výběru vyšší priority vysílané zprávy s nižší hodnotou identifikátoru.

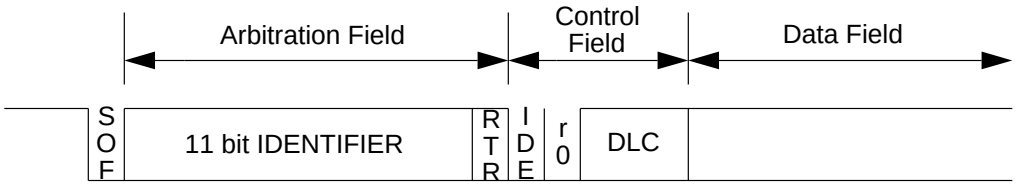
Vlastní datový rámec se skládá z značky začátku rámce, již výše zmíněného identifikátoru zprávy, informace o typu rámce (příznaky RTR a IDE), čtyřbitově zakódovaného počtu datových bytů (DLC), přenášených dat, zabezpečení (CRC) a prostoru pro potvrzení (ACK). Příznak IDE určuje zprávy s 29-bitovým identifikátorem. Příznak RTR informuje příjemce, že zpráva nepřenáší data, ale slouží pouze jako žádost o vyslání dat s odpovídajícím identifikátorem. Zpráva nůže obsahovat až 8 datových bytů. Význam mají i zprávy, které neobsahují žádné datové byty a slouží k přenosu povelů nebo informace o přítomnosti komunikačního uzlu. V prostoru pro potvrzení všechny uzly, které přijmou zprávu a vyhodnotí shodu CRC, vyšlou dominantní bit. Vysílající uzel má tedy potvrzeno, že jeho zpráva byla alespoň jedním uzlem přijata, neznamená to ovšem s jistotou, že zpráva s daným identifikátorem byla přijmuta všemi uzly, nebo že zpráva s daným identifikátorem byla očekávána a zpracována.



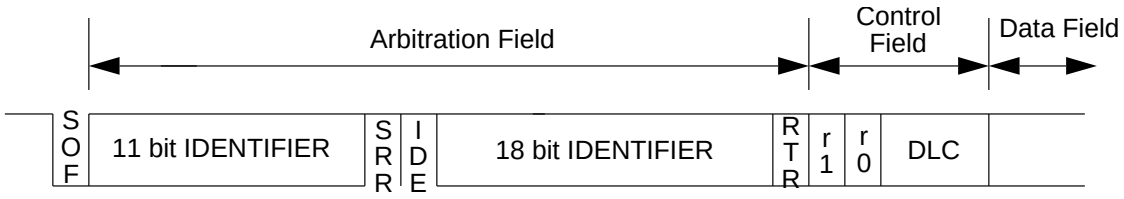
Obrázek 15: Fyzické rozhraní komunikace CAN - rychlá varianta



Obrázek 16: Formát rámce komunikace CAN



Obrázek 17: Standardní hlavička rámce



Obrázek 18: Hlavička s rozšířeným identifikátorem

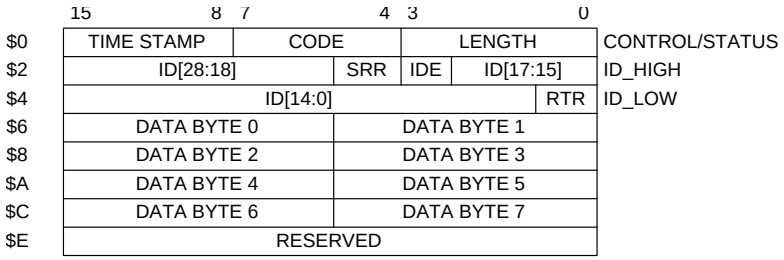
Modul TouCAN implementuje veškerou potřebnou logiku řadiče CAN. Pro implementaci kompletního hardware CAN stačí přidat pouze linkový budič a případné galvanické oddělení. Modul obsahuje subsystém pro nastavení časování přenosu bitů, 16 bufferů pro uložení vysílané nebo přijímané zprávy, logiku pro serializaci a arbitraci přenosů obsahující jeden další zkrytý buffer zprávy a potřebnou řídicí logiku. Každý z 16 komunikačních bufferů býže být použit pro vysílání nebo příjem. Buffer po napnění identifikátorem, daty a označení jako připravený k vysílání přejde po potvrzeném odvysílání opět do stavu neaktivní nebo je-li to požadováno do stavu připravený pro vysílání na vzdálené vyžádání (RTR). Rámec může být vysílaný jako požadavek RTR a po odvysílání přejít do režimu příjmu. Pro příjem je nutné nastavit identifikátor do bufferu, který spolu s globální nebo lokální maskou slouží k filtraci zpráv, které budou do bufferu přijímány. Poté je buffer nastavendo stavu příjmu. Po přijetí zprávy přechází buffer do stavu naplnění. Pokud není do dalšího přijetí zprávy s odpovídajícím identifikátorem buffer vyzvednut procesorem je pro buffer nastaven příznak přeplnění. Modul TouCAN obsahuje tři registry masky, dva lokální pro buffery 14 a 15 a jeden globální pro buffery 0 až 13.

Uspořádání jednoho z bufferů v paměťovém prostoru je znázorněno pro rozšířený formát na obrázku 19. Kromě 8 bytů určených pro data je zde prostor pro uložení identifikátoru (ID0 až 28), příznaků (IDE, RTR a nevyužité kopie RTR z základního rámce SRR), počtu využitých byte (LENGTH) a kódu pro stav bufferu (CODE). Pozice TIME STAMP slouží k uložení vyšší části hodnoty volně běžícího čítače synchronizovaného s přenášenými bity sběrnice CAN. Pro zprávy se základní délkou identifikátoru zbývá v bufferu místo pro uložení celých 16 bitů tohoto čítače a 11-bitový identifikátor je ukládán do oblasti rozšířeného ID18 až 28 a RTR je uloženo na pozici SRR. Uložená hodnota čítače odpovídá okamžiku vysílání nebo příjmu prvního bitu identifikátoru zprávy a může posloužit k přesné časové koordinaci činnosti všech komunikujících uzlů.

Další registry modulu TouCAN slouží pro jeho konfiguraci, povolení generování přerušení od jednotlivých bufferů, monitorování činnosti rozdraní CAN a počítání chyb v příjmu a vysílání zpráv.

4.10 Další mikrokontroléry z rodiny 683xx

Mikrokontroléry z rodiny 683xx se dělí na dvě skupiny, jedna je určena především pro komunikace (například 68360) a v současné době je již nahrazována mikrokontroléry s jádrem ColdFire nebo poverPC. Druhá skupina je určena především pro řídící aplikace (například řízení technologických procesů a robotických systémů). Do této skupiny patří i dříve pop-



Obrázek 19: Formát zprávy s rozšířenou identifikací v bufferu

saný mikrokontrolér 68332. Od jeho uvedení bylo vyvinuto více podobně zaměřených mikrokontrolérů.

- 68331 obsahoval pouze jádro CPU32, integrační modul SIM, modul QSM, 2 výstupy PWM, 3 nebo 4 výstupy komparátorů hodnoty čítače a 4 až 5 záchytů hodnoty čítače
- 68332 obsahuje dříve popsané jádro CPU32 a moduly integrace SIM, komunikace QSM, časovací procesor TPU a 2 kB paměti TPURAM
- 68F333 byl oproti 68332 rozšířen o 10-bitový AD převodník a 64 kB paměti Flash, paměť SRAM byla rozšířena na 4 kB. Mikrokontrolér provázely problémy s technologií výroby paměti Flash což vedlo k jeho stažení z výrobního programu.
- 68334 obsahuje jádro CPU32 a moduly integrace SIM, 10-bitový AD převodník, časovací procesor TPU a 1 kB paměti TPURAM, chybějící modul komunikací lze emulovat na vstupech a výstupech jednotku TPU
- 68336 obsahuje jádro CPU32 a moduly integrace SIM, komunikace QSM, časovačů a čítačů CTM, 10-bitový AD převodník, časovací procesor TPU, 3.5 kB paměti TPURAM a 4 kB paměti SRAM
- 68376 obsahuje všechny moduly mikrokontroléru 68336 a navíc obsahuje modul komunikace CAN a 8 kB paměti ROM pro uložení zavaděče systému

Protože původní mikrokontrolér 68332 byl nejvíce používaný a až na chybějící AD převodník a paměť Flash i nejlépe vybavený, nedošlo k velkému rozšíření ostatních verzí, dokud nebyl uveden mikrokontrolér 68336/376. Tento mikrokontrolér sdružuje téměř vše co může rodina 683xx nabídnout a přidává tolik žádaný AD převodník a modul čítačů a PWM, který je možno pro jednodušší funkce vyžadující velmi přesné časování použít místo TPU. Mikrořadičový TPU není totiž díky proměnné rychlosti odezvy při několika současných požadavcích pro tyto funkce nejvhodnější. V současné době byly již problémy s pamětí FLASH vyřešeny a množství modifikovaných verzí mikrokontroléru 68376 nachází uplatnění v řídicích jednotkách automobilů (například v jednotkách řízení motorů - ECU). Za zmínku stojí nejnovější verze 6837x.

- 68375 nabízí 256 kB paměti FLASH a rozšiřuje paměť TPU na 6 kB, SRAM na 8 kB. Inovuje časovací koprocesor na TPU3 a modul čítačů a časovačů na CTM9. Obsahuje své asynchronní rozhraní (jedno s 16 položkovými FIFO frontami), rozšiřuje autonomní přenosy SPI na 60 položek a QADC na 64 samostatných převodů.

- 68377 je opět rozšířením 68376. Paměť SRAM má kapacitu 32 kB. Další 4 512 bytů dlouhé bloky RAM slouží k překrývání oblastí v paměťovém prostoru. Jsou obsaženy dva moduly TPU3 s emulační RAM, dva moduly QSM, modul QADC s 26 přímými vstupy, časovací modul CTM9 a komunikační kontroléry TouCAN a J1850. Podstatnou změnou je inovované jádro CPU32X, které má 2× větší výkon než CPU32 při shodné frekvenci a může pracovat i na frekvenci 33 MHz.

5 Budoucnost a následníci architektury 68xxx

Při výběru určité rodiny procesorů pro nové aplikace je nutné uvažovat i o jejích předpokladech budoucího vývoje nebo náhrady za výkonnější členy a zachování alespoň částečné kompatibility s již vyvinutým programovým vybavením a aplikacemi. 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 a 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 řídícím aplikacím. Pro aplikace, kde již výkon CPU32 obsaženého například v mikrokontroléru 68332 nepostačuje, lze uvažovat o přidání některého z výkonnějších procesorů (68040, 68060), pro který může být 68332 vstupně/výstupním koprocesorem a nebo pouze vysoce integrovaným blokem periférií. Další alternativou od firmy Motorola jsou mikrokontroléry založené na procesoru PowerPC (například MPC555). Pro oblast komunikací, pro kterou byl původně určen integrovaný čtyřkanálový komunikační procesor 68360, také existují podobné mikrokontroléry založené na architektuře PowerPC (MPC860 a MPC8260).

Vhodnost koncepce rodiny 680x0 pro řídící aplikace vedla výrobce k úvahám o jejích dalších výkonnějších nástupcích. Výsledkem jsou mikrokontroléry založené na procesorovém jádře ColdFire 52xx [4], jehož architektura je založena na 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. Jádro čtvrté generace procesorů ColdFire 54xx již dosahuje výkonu až **257 MIPS** @ 162 MHz.

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 mohou být rozdíly v integrovaných perifériích (kromě kombinace 68060 a 68332, kde periférie zůstávají) a v návrhu nového hardware.

Protože se velké množství integrovaných subsystému vyvinutých pro mikrokontroléry 683xx osvědčilo a stalo se vzorem i pro konkurenční výrobce, lze nalézt většinu těchto subsystémů naintegrovaných i na mikrokontrolérech založených na jádře PowerPC.

5.1 Architektura PowerPC

Architektura vychází z procesoru POWER1 implementovaného původně na třech čipech pro výkonné počítače IBM RS6000. Procesor sice přebírá množství prvků procesorů RISC a odpovídá load-store architektuře. Jeho realizace je však velmi složitá a obsahuje velké množství instrukcí, které jsou vždy kódovány do jednoho 32 bitového slova. Návrh integrovaného procesoru PowerPC vznikl v roce 1992 ze spolupráce společností IBM a Motorola jako protiváha procesorů Intel. Procesory PowerPC se skládají z několika co nejvíce nezávislých funkčních jednotek.

Jednotka skoků obsahuje čítač instrukcí a registr příznaků CR a obstarává načítaní instrukcí, jejich předdekódování a s nulovým zpožděním řeší většinu skoků. V novějších implementacích je rozdělena na samostatné jednotky načítání instrukcí a řešení cílů skoků, koordinační jednotku pro zasílání instrukcí do ostatních funkčních jednotek a jednotku kompletace mimo pořadí vyhodnocovaných výsledků a výjimek. Příznakový registr je rozdělen na 8 čtyřbitových polí CRx. Pole CR0 může být volitelně nastavováno aritmetickými a logickými instrukcemi, pole CR1 je implicitně nastavováno operacemi v plovoucí řádové čárce a pole CR7 je rezervováno pro vektorový koprocesor. Instrukce komparací a skoků mohou pracovat s libovolným polem. Jednotka skoků obstarává i volání podprogramů s možností uložením adresy následující instrukce do návratového registru (LR viz zmínka o leaf-node funkcích). Pro urychlení implementace smyček je jednotce implementován čítač smyček (CTR). V novějších implementacích byla statická predikce skoků uložená v instrukčním kódu rozšířena i o predikci na základě historie skoků.

Dále procesory obsahují jednotku přesunů, jednu až tři jednotky pro operace v pevné řádové čárce a jednu jednotku pro operace v plovoucí řádové (dvě v případě architektury POWER2). Většinu instrukcí kromě násobení a dělení vykonává procesor v jednom cyklu. Tříoperandové aritmetické a logické operace pracují s 32 obecnými registry (GPRs - general purpose registers). Registr GPR0 slouží v některých operacích jako zdroj nulové hodnoty. Registry se v kódu označují r0 až r31. Pokud jsou implementovány, provádějí se operace v plovoucí řádové čárce mezi 32 k tomu určenými 64 bitovými registry (FPRs). Pro řízení FPU a obsloužení výjimek podle normy IEEE je FPU doplněna řídicím a stavovým registrem FPSCR. Architektura POWER definuje jako nejvíce významný bit slova (MSB) bit označený 0, nejméně významný bit 31. Architektura umožňuje činnost procesoru jak v režimu big-endian, tak v režimu little-endian.

Procesor přenáší data mezi registry a pamětí jen k tomu určenými instrukcemi. Množství adresních módů je omezené. Adresa může být určena jedním nebo dvěma registry (rA|0+rB) a nebo kombinací registru a 16-ti bitového znaménkově rozšířeného ofsetu (rA|0+d). Hodnota 0 je určena kombinací určenou pro GPR0. Nepodmíněné skoky využívají 26 bitový nebo 16 bitový znaménkově rozšířený offset.

Pro usnadnění přenosu aplikací mezi různými implementacemi a generacemi procesorů

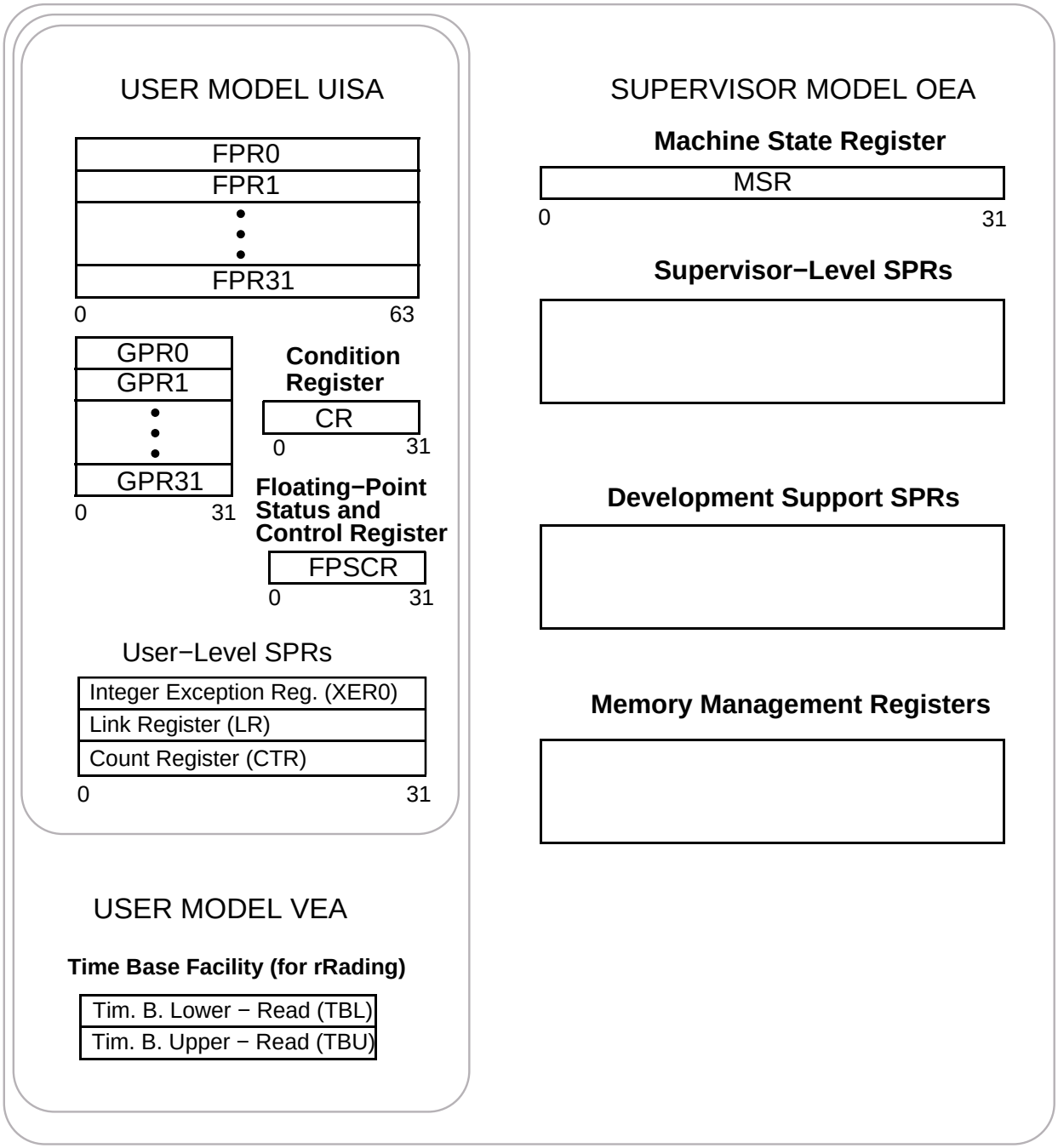
byly definovány tři úrovně architektury jak je vidět na obrázku 20. Základní uživatelská úroveň UISA (user instruction set architecture) je pro účely spolupráce programů v multiprocesním a multiprocesorovém prostředí rozšířena o instrukce a speciální registry (SPRs) virtuální úrovně (VEA - virtual environment architecture). Pro jednodušší mikrokontroléry MPC566 je například implementován pouze registr časové základny. V systémovém režimu je pro řízení procesoru a případnou zprávu paměti může být implementováno množství dalších registrů. Tato systémová úroveň je označována OEA (operating environment architecture). Pokud implementace obsahuje správu virtuální paměti, není většinou implementována celá hardwarově a o plnění vyrovnávacích adresových překladových tabulek se musí starat kód operačního systému.

5.2 Mikrokontroléry na bázi PowerPC

Společnost Motorola nabízí dvě hlavní skupiny mikrokontrolérů založených na architektuře PowerPC. První skupina obsahující mikrokontroléry MPC555/6 a MPC565/6 je určena především pro řídicí aplikace. Kromě procesorového jádra s jednotkou pro operace v plovoucí řádové čárce a bez jednotky zprávy paměti jsou na čipu integrovány rozšířené a často zdvojené verze všech modulů popisovaných u mikrokontroléru 68376. Napájení je kombinované 5 V pro vstupy a výstupy 3.3 V nebo 2.6 V pro interní logiku. Mikrokontroléry obsahují až 1 MB paměti FLASH a 50 kB paměti RAM v několika blocích. Jako všechny mikrokontroléry Motorola pro řídicí aplikace jsou dostupné v průmyslových i vojenských teplotních (-40 až 125°C) rozsazích a nacházejí uplatnění v nepříznivých podmínkách (například pro řízení automobilových motorů i vozů Formule-1).

Druhou skupinou jsou mikrokontroléry určené pro komunikační aplikace. Vycházejí z procesoru PowerPC 603e přepracovaného na mikrokontrolér MPC823 následovaný celou rodinou příbuzných mikrokontrolérů. Obsahují až 16 kB instrukční a 8 kB datové paměti cache a skladbou periferií vycházejí s mikrokontrolérů 68360. Například mikrokontrolér MPC860 (PowerQUICC) může realizovat až 4 rozhraní Ethernet 10T, jedno rozhraní Ethernet 10/100, obsahuje čtyřkanálový programovatelný sériový koprocesor a možnost komunikace ATM a T1/E1.

Pro vysoce výkonné komunikační aplikace je připravována rodina mikrokontrolérů MPC825x/6x označovaná jako PowerQUICC II. Integruje až 16+16 kB paměti cache, 4 rozhraní Ethernet 10T, 2 rozhraní Ethernet 10/100, dva kanály ATM a množství dalších sériových kanálů a protokolů.



Obrázek 20: Úrovně architektury procesorů PowerPC

6 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.

6.1 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 nadřízeným 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.

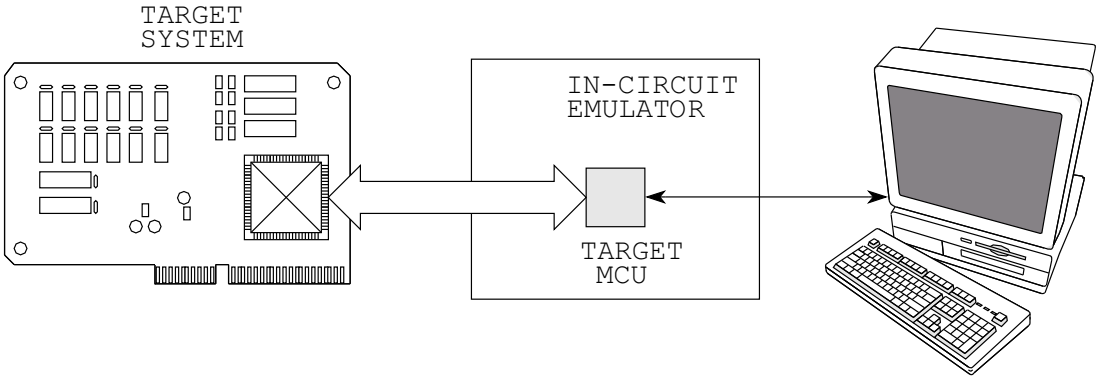
Protože je integrace dnešních procesorových obvodů již natolik vysoká a interní mikrořadič dostatečně výkonný, je možné přidat do mikroprogramů funkce umožňující při vývoji komunikovat přímo s jádrem mikroprocesoru. Takové řešení sice mírně zvyšuje složitost vyráběného obvodu, ale není pak nutné vyvíjet 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 nadřízeného 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 21) a systém využívající interního emulátoru procesoru MC68332 (Obrázek 22. 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 nadřízeného počítače, může být v mnoha případech rychlejší, než komunikace s klasickým externím emulátorem přes interface RS232.

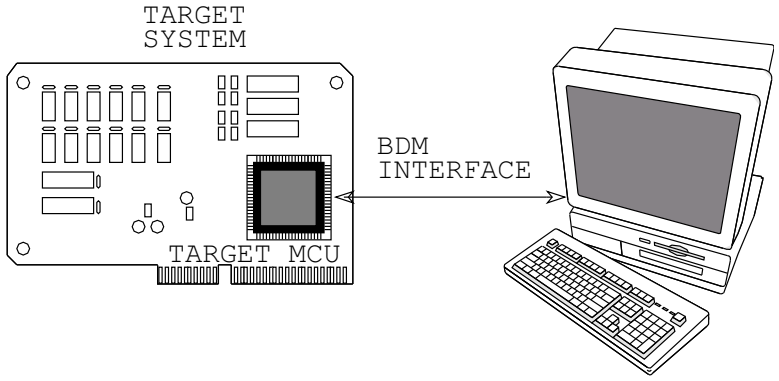
6.2 Prostředky pro vývoj software

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 - assembler a strojový kód).



Obrázek 21: Externí emulátor



Obrázek 22: Interní BDM emulátor

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 platforem. 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.

6.3 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 řídícím 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 nadřízeného 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.

7 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.

7.1 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 **UNIX**u. 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.

7.2 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 **DOS**u (**DJGPP**).

7.3 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.

7.3.1 Příprava instalace balíku BINUTILS

Balík se zdrojovými texty výše zmiňovaných programů je možné získat na velkém množství FTP servrů pod názvem **binutils-2.7.0.3.tar.gz** pro popisovanou 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 7.3.7.

7.3.2 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 dostačně 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.

7.3.3 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 Format[2]

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

[2]Čtenář si pravděpodobně pomyslí, že se nejedná o nic zvláštníno v porovnání s formátem dynamických knihoven DLL v systému Windows. Ovšem Windows natahují celý program i knihovny do paměti a provedou relokaci přímo změnami 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 textový stránlky bez nutnosti jejich zápisu do odkládacího souboru, protože nejsou modifikované a je možné je v budoucnosti načíst znovu z mapovaného spustitelného souboru.

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 7.3.6.

7.3.4 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 7.3.1.

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

7.3.5 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 platforem do jednoho adresáře. Pak je možné programy odlišit pomocí předpony a přípony přímo ve jménu programu (*program-prefix* a *program-sufix*).

7.3.6 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 interpretr Berkley Shell systému UNIX. Spouští se příkazem *./configure*[3], 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]

[3]znaky tečka a lomeno "./" před jméne programu jsou nutné, protože většina instalací **UNIX**u 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.

–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]

–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 MC688332 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

7.3.7 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 platforem, jsou teoreticky pro přidání další platformy specifické pouze soubory *as* a *ld*. Vše ostatní může být nahrazeno symbolickými linkami.

7.3.8 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 řádku 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 7.3.3) se vyžádá přepínačem *-pic*. Pro překlad souborů s originální (Motorola) syntaxí instrukcí (nikoli AT&T a UNIX) je možno použít přepínač *-M*.

7.3.9 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 souboru (***.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 *-oformat* . 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*.

7.3.10 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.

7.3.11 BINUTILS - další programy

objdump výpis obsahu 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

7.4 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 řídících 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**.

7.4.1 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 servrů 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.

7.4.2 Parametry programu configure pro GCC

Program je opět soubor skriptů pro interpretr Berkley Shell systému UNIX. Spouští se příkazem *./configure*, který je následován požadovanými parametry. Způsob zadávání jmen platforem je shodný s balíkem **BINUTILS** viz 7.3.4. 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

7.4.3 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 definicí 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 x*m-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 zkompilovaným 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*.

7.4.4 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 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*.

7.4.5 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 řídící 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 7.4.2)

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

V době kompilace by již měly být nainstalovány programy z balíku BINUTILS (viz 7.3) 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 čárce 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 kompileru

```
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 řídící systémy možný, pak je nutné použít již předdefinovaný hlavičkový soubor *float.h*. Tento soubor použitelný pro většinu cílových platforem je k dispozici z cross-gcc distribuce firmy CYGNUS.

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

7.4.6 GCC - instalace crosscompileru

Nejjednodušší je spustit instalaci příkazem

```
make install
```

Je však vhodné vědět, do kterých adresářů jsou uloženy jednotlivé programy a soubory kompileru. 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) kompiléru 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

8 BDM vývojové rozhraní a GDB

Popis zatím existuje pouze v anglické verzi na stránce `http://cmp.felk.cvut.cz/~pisa`.
konex 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] RCPU RISC 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

[9] Motorola ColdFire Development Resources by David Fiddes:
`http://fiddes.net/coldfire/`

[10] BDM Interface for Motorola 683xx MCU (Usage with GDB Debugger)
Pavel Píša, Praha, 1999
`http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.html` HTML formát
`http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.pdf` PDF formát

Obsah

Seznam obrázků