Processzorok

Processzor adatstruktúrák vizsgálata

Az általános célú adatfeldolgozó egységek milyen lehetséges típustervek alapján épülhetnek fel? Mi az utasításformátumok jellemzője (méret, tartalom, rugalmasság)?

Röviden: A feldolgozóegység bemeneteinek és kimenetének címzésétől függően létezik nulla-, egy-, kettő- és háromcímes utasításformátum, mely ebben a sorrendben egyre rugalmasabban alkalmazható.

Adatfeldolgozó egységek lehetséges típustervei:

- Verem alapú feldolgozás: az aritmetikai-logikai egység regiszterei veremmemóriát képeznek, melynek két legfölső regisztere adja mindig a feldolgozóegység bemeneteit, az eredmény pedig a verem tetejére kerül. Ebben az esetben, a be- és kimenetek fix elhelyezkedése miatt az általános utasításformátum csak az elvégzendő művelet kódját (OPCODE) tartalmazza. Ez a felépítmény rendkívül rövid utasításformátumot eredményez, ami memória-hatékony, azonban az adatmozgatás tekintetében rugalmatlan.
- Akkumulátor alapú feldolgozás: az aritmetikai-logikai egység rendelkezik egy regisztertömbbel és benne egy kiemelt fontosságú "akkumulátor" regisztert. A feldolgozóegység egyik bemenete mindig az akkumulátor, valamint az eredmény is ebbe kerül elhelyezésre, a másik operandus (regiszter) címe az utasításban kerül megadásra, az utasítás kódja mellett. Az elrendezés továbbra is relatíve rövid utasításformátumot eredményez, a bemenetek kezelésében pedig rugalmasabb, azonban az, hogy minden adatot az akkumulátoron keresztül lehet csak mozgatni a program futása közben, bizonyos mértékű rugalmatlanságot jelent.
- Két regiszteres utasításformátumú feldolgozás: a műveletvégző mindkét operandusának regisztercímét meg kell adni, az eredmény pedig egy előre meghatározott ("eredmény-") regiszterbe kerül. Ez a típus a bemeneteket tekintve rugalmasabb, mint az akkumulátor alapú megoldás, azonban a kimenet továbbra is fix, ami plusz adatmozgatást/adatmentést igényelhet, miközben az utasításformátum hosszabbá vált.
- Három regiszteres utasításformátumú feldolgozás: a műveletvégzőnek mindkét bemenete és kimenete is megadható, ebből kifolyólag ez a legrugalmasabban alkalmazható felépítmény, de az utasításformátum a regisztertömb mélységétől függően meglehetősen széles is lehet.

Mi a jellemzője egy algoritmus/számítási feladat közvetlen hardver megvalósításának? (művelet/órajel, felépítés/struktúra, végrehajtás vezérlése)?

Előnyök:

- A végrehajtó egység felépítése az algoritmusra szabható, ami a lehető leggyorsabb végrehajtást eredményezi, mely egyszerű számítások vagy pipeline-osítás esetén minden órajelben új eredményt szolgáltathat.

- Jól párhuzamosítható algoritmusok esetén nagyobb mértékű lehet a párhuzamosítás hardverben, mint szoftverben.
- Adatfolyam jellegű feldolgozás esetén, amikor minden "automatikusan" az órajelre történik, a hardveres megvalósítás nem igényel komoly vagy akár semmilyen vezérlést.

Hátrányok:

- Egy adott algoritmusra szabott hardver nem képes más algoritmus végrehajtására.
- Nagy mértékben szekvenciális algoritmus esetén a vezérlő logikai összemérhetővé válik a lényegi számításokat végző logikával, szélsőséges esetben a terv célhardverből általános processzorrá fajul.

Algoritmusok hardveres megvalósítása esetén költség alatt (FPGA-ban) a felhasznált műveletvégző egységek számát vagy (ASIC-ban) a feldolgozóegység(ek) szilícium-terület-igényét értjük.

Mi a jellemzője az általános célú műveletvégzőn, szoftver vezérlés mellett megvalósított algoritmus-végrehajtásnak? Mit értünk ezekben az esetekben egy program költségének? Mi alapján adódik az optimum?

Előnyök:

- A szekvenciális programvégrehajtásból adódóan a szekvenciális algoritmusok esetén előnyös megoldást jelentenek.
- Általános célú műveletvégzővel tetszőleges algoritmus megvalósítható.
- Az műveletvégző programjának megváltoztatásával (átprogramozással) könnyedén változtatható a végrehajtandó feladat.

Hátrányok:

- Az utasítás- és adatbetöltés valamint adatmentés szükséges lépései miatt az általános célú műveletvégzők nem tudnak minden órajelben új eredményt szolgáltatni.
- Az algoritmust (programkód formájában) memóriában kell tárolni, ami többlet terület-igényt jelent a szilíciumon vagy a nyomtatott áramköri lapon.

Általános célú processzoros implementáció esetén az algoritmus vagy program költsége alatt a szükséges programmemória méretét értjük. A különböző típusú, általános műveletvégzők mind más utasításszélességet igényelnek, azonban a rövidebb utasítások nem feltétlenül jelentenek kisebb költséget: rugalmatlan utasításokból nehézkes, hosszadalmas programkód lesz, ami összességében akár hosszabb is lehet annál, mint amit szélesebb utasításokkal kapnánk. Az optimum ott van, ahol az egyes utasítások nem túl hosszúak, de már megfelelően rugalmasak és kifejezőképesek.

RISC-V

Mi határozza meg egy CPU teljesítményét?

Röviden: 1) Utasításarchitektúra: hány darab és milyen bonyolultságú utasítás alkotja (befolyásolja a létrejövő gépi kód hosszát). 2) Utasítás-végrehajtási sebesség: órajel-frekvencia szorozva az órajelenként végrehajtott utasítások számával.

Egy CPU teljesítményét két tényező befolyásolja: hogy hány utasítással rendelkezik, illetve hogy ezeket milyen gyorsan képes végrehajtani.

Az utasítások darabszáma az utasítás architektúrából (Instruction Set Architecture, ISA) következik, melynek két főbb irányvonala létezik: a RISC és a CISC. RISC (Reduced Instruction Set Computer) esetén az utasításkészlet viszonylag kis számú (30-50) és egyszerű (elemi aritmetika, logika, adatmozgatás) utasítást tartalmaz, míg CISC (Complex Instruction Set Computer) esetén nagy számú (több száz) és nagy bonyolultságú utasítást. Előbbi esetben a fordító által generált gépi kód sok-sok elemi utasításból fog állni, utóbbi esetben pedig mindössze néhány speciális, összetett utasításból.

Az utasítások végrehajtási sebessége két dologtól függ: az órajel-ciklusonként végrehajtott utasítások számától, illetve magától az órajel-ciklus hosszától (órajelfrekvenciától) Míg az órajel-frekvencia tisztán hardveres kérdés, az utasítások órajel-periódusban számolt hossza az utasítás-architektúrával összefüggő tényező: RISC architektúra esetén az egyszerű, elemi utasítások jellemzően egyetlen órajel alatt végrehajthatók, addig az összetett utasításokat alkalmazó CISC ISA esetén egyetlen utasítás több órajel-ciklust is igényelhet.

Mi a szerepe a 2 ADD jelű egységnek a program végrehajtás szempontjából a 4. dián?

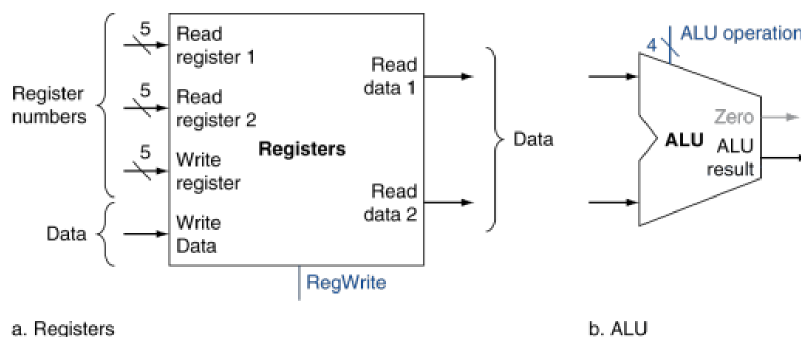
A soron következő utasítás címét állítják elő. A bal oldali összeadó a hivatott a jelenlegi utasítás címéhez 4-et hozzáadva állítja elő a következő utasítás címét (a 4-es szám a 4 bájtos, 32 bites utasításméretből következik). A jobb oldali összeadó nem szekvenciális programvégrehajtás (ugró utasítások) esetén aktív, ekkor az ugró utasításban található relatív címet hozzáadja a jelenlegi címhez.

Mik a fontosabb részmodulja a CPU adatstruktúrának?

- A Program Counter és az azt meghatározó utasításcím-aritmetika,
- az utasításmemória,
- a műveletvégző egység és a hozzátartozó regisztertömb,
- az adatmemória.

Rajzolja le egy 3 operandusú utasításarchitektúrájú CPU regisztertömbjének szimbólumát és definiálja az interfészeit 32 db regiszter használatához (címek, adat ki/bemenetek, vezérlés)!

A regisztertömbnek - 3 operandus esetén - van három darab címbemenete, melyek szélessége ldN (N=32 esetén 5), valamint maga az adatbemenet és a hozzátartozó írásengedélyező jel; kimenetből kettő van, melyeken a megcímzett két operandus kerül kiolvasásra.



1. ábra. 3 operandusú műveletvégző regisztertömbje és ALU egysége

Ismertesse a RISC-V utasítás-végrehajtás pipe-line fokozatait!

A RISC-V architektúra ötfokozatú utasítás-végrehajtásának lépései:

1. utasítás beolvasása,
2. utasítás dekódolása és regisztercímzés,
3. műveletvégzés és címszámítás,
4. az eredmény adatmemóriába való elmentése,
5. az eredmény memóriából regiszterbe történő visszaírása.

Miért érdemes pipe-line utasítás-végrehajtást alkalmazni?

Bár maga az utasításban foglalt művelet végrehajtása egyetlen órajel alatt megtörténik, az utasítást megelőző és azt követő "adminisztráció" alacsony hatékonyságot eredményez. A teljes utasítás-végrehajtási folyamatnak csak kis részében vannak kihasználva az egyes fokozatok egységei (kiemelendő ezek közül maga az ALU egység). A programvégrehajtás sebességének növelése és az erőforrások jobb kihasználtságának érdekében a fokozatok közé tárolókat (pipeline regiszterek) helyezhetünk, hogy így az egyes fokozatok párhuzamosan dolgozhassanak és az utasítások végrehajtása átlapolódhasson.

Mikor van probléma a pipe-line utasítás-végrehajtással?

A pipeline ellenségei: ugró utasítások, még ki nem számolt eredményt használó utasítások, erőforráshiány.

A pipeline-osítás akkor hatékony, ha a fokozatok egymástól függetlenül képesek dolgozni. Ez nem igaz abban az esetben, amikor ugró utasítás kerül a "futószalagra", ugyanis ekkor nem tudhatjuk egészen az ugrási feltétel kiértékeléséig (harmadik fokozat), hogy az utasítás beolvasó szintnek (első fokozat) a soron következő vagy valahol teljesen máshol lévő utasítást kell beolvasnia. Ha az ugrási feltétel igaznak értékelődik ki, akkor az addig beolvasott és dekódolt soron következő utasítások eldobásra kerülnek, a címszámítás után pedig újra be kell olvasni az ugrási címen lévő kódot: ugró utasítások tehát részben vagy egészben kiüríthetik a pipeline-t, melyet a hatékony működés érdekében újra fel kell tölteni.

Az ugró utasításokon kívül gondot okoz még az, ha a soron következő utasítás egy korábbi, még nem kiértékelt utasítás eredményét szeretné használni. Ekkor az utasítás-végrehajtás szintén szünetel, ameddig a szükséges eredmény elő nem áll.

Gondot okozhat még az is, ha az utasítás végrehajtásához olyan speciális egység szükségeltetik, amely jelenleg elfoglalt, így nem áll rendelkezésre, vagy maga az utasításfelhozatal, operandus beolvasás, eredménykiírás, mint memóriaműveletek ütköznek, ennek elkerülése érdekében Harvard-architektúra vagy külön utasítás- és adat cache alkalmazása célszerű.

Milyen utasítás-végrehajtási hazárdok fordul(hat)nak elő?

- Strukturális hazárd: az utasításokkal és az adatokkal kapcsolatos memóriaműveletek ütközhetnek, a várakozás "buborékot" hozhat létre a pipeline végrehajtásban. A konkurens memória-hozzáférések kiküszöbölhetők külön utasítás- és adatmemória (Harvard-architektúra) vagy külön utasítás- és adat cache alkalmazásával.

- Adat hazárd: a soron következő utasítás egy vagy több operandusa nem áll még rendelkezésre, mert az(ok) még ki nem értékelt korábbi utasítások eredményei, vagy már kiértékelődtek, de nem íródtak vissza a memóriába, ahonnan beolvashatnánk. Az adat hazárd kiküszöbölhető hardveres és szoftveres támogatással: hardveres esetben a műveletvégző kimenete közvetlenül visszacsatolható annak valamely bemenetére, hogy a

memóriába való visszaírás és az onnan újra kiolvasásának idejét megspóroljuk; szoftveres esetben a magas szintű programnyelv fordítója a gépi kód előállítása során figyeli, hogy az eredményt felhasználó utasítás ne legyen túl közel az eredményt előállító utasításhoz, ennek érdekében pedig megváltoztatja az utasítások sorrendjét, természetesen csak az eredeti program szemantikai helyességét még nem csorbító mértékben, független utasítások cseréjével.

- Vezérlési hazárd: a feltételes ugró utasítások kiértékeléséig nem ismert, hogy vajon melyik utasítást kell a következő ütemben beolvasni. A problémával nem törődve előállhat, hogy a feltétel igaznak értékelődik ki és a már beolvasott, dekódolt utasításokat eldobni kényszerülünk, a pipeline ekkor (részben) kiürül és újratöltődik. A vezérlési hazárd elkerülésének módszere a feltétel korai kiértékelése, ez azonban hosszú pipeline struktúra esetén nem lehetséges, ezért előrejelzést alkalmaznak. RISC-V esetén az előrejelzés azt feltételezi, hogy a feltétel hamissá fog kiértékelődni és a program a memóriában következő utasítással folytatódik. Az előrejelzésnek létezik statikus és dinamikus fajtája is, előbbi az elágazások "tipikus" viselkedését veszi alapul, utóbbi a működés során statisztikát tart fenn az eddig kiértékelt feltételek eredményéről, ezekből "trendeket" feltételez és annak megfelelően vagy hamisnak, vagy igaznak feltételezi a soron következő ugrófeltételeket.

Mit jelent a dinamikus ugrás becslés az utasítás-végrehajtás szempontjából?

A dinamikus ugrásbecslés a feltételes ugró utasítások kiértékelés előtti "előrejelzését" takarja, az előrejelzéstől függően folytatódik a pipeline feltöltése a memóriában következő vagy az ugrási címen lévő utasítással. Az előrejelzésnek statikus és dinamikus verziója is létezik, dinamikus esetben hardveresen nyilvántartást vezet a feldolgozóegység a múltban kiértékelt ugrási feltételek eredményéről és ezekből "trendeket" levonva ítéli a következő feltételeket igaznak vagy hamisnak. Az elágazás előrejelzés dinamikus verziója bonyolultabb és erőforrás-igényesebb, mint a statikus megoldás, azonban ez adaptív előrejelzést biztosít így hatékonyabb is, ami hosszú pipeline struktúra esetén kifizetődő.

Mi a kivétel és mi a megszakítás? Hogyan kezelhetők?

A kivétel és a megszakítás egyaránt a normál programfutást kizökkentő események, okuk azonban különböző. Megszakítást jellemzően perifériától, I/O egységtől kap a processzor, míg kivételek hibás programműködés, ismeretlen OPCODE dekódolási kísérlete esetén vagy hardveres meghibásodás alkalmával adódik.

Mindkét esetben fontos a processzor kizökkenését megelőző állapot elmentése, hogy a kivétel- vagy megszakításkezelést követően visszatérhessen a program futása a normális kerékvágásba. Megszakítás esetén a pipeline kiürítésre kerül és a megszakítás forrásának felderítése után a megfelelő megszakítási rutin kerül meghívásra. Kivételkezelés esetén is hasonló az eljárás, a pipeline-ban lévő, félig feldolgozott utasítások "kisöprésre" kerülnek, a kivétel okát pedig megfelelő regiszterekbe mentjük és meghívjuk a kivételkezelést. RISC-V esetén ezek a regiszterek a SEPC (Supervisor Exception Program Counter, a PC elmentéséhez) valamint az SCAUSE (Supervisor Exception Cause Register, a kivétel fellépésének okát jelezendő). Hosszú pipeline struktúrák esetén egyszerre több kivétel is keletkezhet, ebben az esetben kezelhetjük az első vagy akár az összes kivételt, utóbbi természetesen költségesebb megoldás.

Tensilica Xtensa

Mit jelent a konfigurálhatóság a hatékonyság szempontjából? Miért van értelme használni?

A hatékonyság záloga ha minél jobban az algoritmusra szabott műveletvégzővel rendelkezünk, ekkor azonban a műveletvégző veszít rugalmasságából. A flexibilitás és a hatékonyság együttes

megvalósításának akkor áll fenn a lehetősége, ha a műveletvégző (át)konfigurálható a feladattól függően.

Milyen paraméterek/tulajdonságok konfigurálhatóak [az Xtensa esetében]?

Az Xtensa általános célú processzor(sablon), mely tervezési időben az adott alkalmazásra szabható a hatékonyság érdekében. Bizonyos paraméterek megváltoztathatóak (utasításkészlet, memóriaszervezés, interfészek, perifériák), bizonyos elemek kivehetők (méret- és fogyasztáscsökkentés érdekében), meglévő elemek többszörözhetők vagy újak beépíthetők (saját IP).

Milyen a Tensilica utasításarchitektúrája?

Az Xtensa utasításkészlete alaputasításokra és felhasználó által definiált utasításokra oszlik. Az alap utasítások 16-24 bit szélesek, ezzel szemben a felhasználó akár 128 bit széles utasítást is definiálhat. Az utasításarchitektúra rugalmasságát támogatja továbbá, hogy 2-30 darab különféle feldolgozóegység is beépíthető. Az utasításokat, független a méretüktől vagy hogy alaputasítások-e, egységesen, "overhead" nélkül kezeli az utasításértelmező.

Mit jelent a VLIW utasításformátum? Miért kell a VLIW utasításformátum?

A Very-Long-Instruction-Word utasításformátum olyan esetben kerül előtérbe, amikor egy speciális, jellemzően bonyolult utasítássorozatot szeretnénk egyetlen gépi utasításba összevonni, az elnevezés pedig abból adódik, hogy ilyenkor a bonyolult utasításkód és az operandusok nagy száma a szokásosnál sokkal hosszabb utasításformátumot eredményez. Mivel alkalmazásspecifikus utasításokat akkor érdemes bevezetni, amikor egy gyakran ismételt műveletsort szeretnénk (kevesebb gépi utasítás felhasználásával) kevesebb órajel alatt elvégezni, ezért az ilyen összetett utasítások általában VLIW kategóriába esnek. Jellemzően VLIW-nek számítanak a SIMD-(Single Instruction Multiple Data) vagy vektorutasítások is.

Hogyan definiálható a Tensilica Instruction Extension? Említsen egyszerű példákat összevont utasításokra, SIMD működtetésre, párhuzamos végrehajtásra!

A Tensilica Instruction Extension koncepciója, hogy a processzor adatfeldolgozó egységét (és csak ezt) transzparens módon kiegészítjük egyéni feldolgozó elemekkel. A kiegészítések specifikálására a Veriloghoz hasonló TIE leírónyelven történik: az *operation* kulcsszó után következik az új utasítás neve, majd felsorolandók az utasításformátum elemei (ki- és bemeneti regiszterek neve, mint egy függvénydeklarációnál), majd az operáció leírása következik.

Példák:

- Átlagolás utasítás megvalósítása összeadás és shift-elés összeolvasztásával: "(a + b) » 1" egy utasításban.
- Vektorok összegzése tipikus SIMD művelet: az összeadások egymástól függetlenül, azonos utasításkóddal elvégezhetők (párhuzamosítás is szükséges).
- Műveletigényes, de jól párhuzamosítható alkalmazások: Viterbi-kódolás, MPEG4 mozgásbecslés.

Mit jelent a CPU közvetlen bekapcsolása az adatáramlási láncba? Mi az előnye ennek a megoldásnak?

Az általános célú processzorok a legtöbb jelfeldolgozási feladatot csak több lépésben, több utasítás egymás utáni elvégzésével képesek elvégezni, ezért nehezen vagy egyáltalán nem illeszthetők be adatáramlási láncba. Ha a processzort speciális, összevont illetve SIMD utasításokkal egészítjük

ki, elegendően felgyorsítható ahhoz, hogy közvetlenül bekapcsolódhasson a jelfolyamba. Az adatáramlási láncba való bekapcsolódás előnye, hogy megkerülhető a rendszerbuszon keresztüli adatmozgatás memóriából illetve memóriába.

Foglalja össze a Tensilica konfigurálható technológia jellemzőit, előnyeit!

Az Xtensa processzorral való tervezés jellemzői:

- Rendelkezésre áll egy közel kész "processzorsablon" a fejlesztés megkezdéséhez.
- A processzor műveletvégző egysége és interfészei tervezési időben opcionálisan ki/behelyezhetők, konfigurálhatók, előbbi alkalmazásspecifikus blokkokkal kiegészíthető, a kiegészítések transzparens módon épülnek be az architektúrába.
- Az alkalmazásra szabott processzorhoz automatikusan generálódik magas szintű fordító, amely megkönnyíti az egyedi architektúrára való szoftverfejlesztést. A fordítón kívül az alacsony szintű vezérlést elfedő firmware is generálódik.

A Tensilica technológia alkalmazásának előnyei:

- A processzor tervezési időben konfigurálható, az alkalmazásra szabható.
- A "félkész processzorsablonnak" köszönhetően nem kell teljesen az alapoktól megtervezni a processzort, ami nagy mértékben csökkenti a szükséges anyagi- és emberi erőforrások valamint a szükséges fejlesztési idő mértékét.
- A jelfeldolgozási képességei plusz műveletvégzők beépítésével, párhuzamosítással jelentősen javíthatók, miközben megmarad a szoftveres programozhatóság által biztosított rugalmasság és egyszerű, kényelmes alkalmazásfejlesztés.

A Tensilica technológia alkalmazása azonban, mint ahogy az az ASIC tervezésre általában is igaz, csak nagy mennyiségű gyártás és forgalmazás esetén gazdaságos.