Általános INFORMATIKAI Tanszék

Számítógépek, számítógép rendszerek

5. Hardver architektúrák, a CPU

Dr. Vadász Dénes

Miskolc, 2005. február

TARTALOM

5. Hardver architektúrák, a központi egység működése

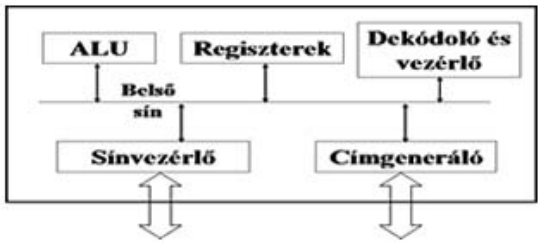
Láttuk az 1.1. ábrán az általános (Neumann elvből következő) architektúrát. Ezek szerint egy számítógép fő hardver komponensei - megemlítve a feladataikat is - a következők:

- A központi egység (CPU: Central Processing Unit). A CPU az általános vezérlő, műveletvégző és adatmozgató egység.
- A központi tár (Central Memory), a memória. A programok és az adatok tárolására szolgál.
- A sín, sínek (bus). A sínek adatmozgatást biztosító áramkörök.
- Az I/O perifériák, eszközök (device). Feladatuk a "másodlagos", "harmadlagos" tárolás és a külvilággal való kapcsolattartás.

A *számítógép működése* nagyon leegyszerűsítve és általánosan a következő:

A CPU "veszi" a tárból a soron következő gépi utasítást (Machine Instruction) és az esetlegesen szükséges adatokat. Elemzi az instrukciót és végrehajtja. Eredményét a CPU-ban tartja, vagy mozgatja a memóriába, majd folytatja a működését a soron következő instrukció feldolgozásával. Egyes instrukciók képesek a perifériákat kezelni, ugyanakkor egyes perifériák meglehetős önállósággal is képesek működni. A működés összehangolása a megszakításrendszeren keresztül biztosított.

A következőkben kicsit részletesebben vizsgáljuk az általános strukturális elemeket. Először a CPU-t, annak architektúráját, működését vesszük.



5.1. ábra. Egy CPU architektúra

Manapság a CPU rendszerint egy mikroprocesszor, ami egy nyomtatott áramköri lapon van. Lehet persze a CPU maga különböző aktív és passzív elektronikai elemekből álló nyomtatott áramkör is. A funkcionális felépítése nagyon általánosan a következő (5.1. ábra):

Nézzük a CPU általános strukturális elemeit.

5.1. Az ALU (Aritmetikai logikai egység)

Ha úgy tetszik, ez a CPU - egyben a számítógép - "kalkulátora", ami néhány alapvető műveletet képes végrehajtani.

- Összeadás és kivonás. Kezeli a helyiérték átviteli biteket.
- Fixpontos szorzás és osztás.
- Léptetések (shift), bitek mozgatása jobbra/balra (ami már a fixpontos szorzás/osztáshoz úgyis kell).
- Lebegőpontos aritmetikai műveletek. Ezeket nem minden ALU képes elvégezni. Néha a processzoron kívül, néha azon belül külön komponens végzi ezeket a műveleteket.
- Egyszerű logikai műveleteket.

5.2. A regiszterek, regiszterkészlet

A regiszterek a CPU belső tároló elemei. Tartalmuk gyorsan (a leggyorsabban) és egyszerűen elérhetők a CPU elemei (ALU, dekódoló, stb.) számára. „Munkamemóriát" biztosítanak az ALU számára, ideiglenes tárolást biztosítanak, segítik a címképzést, tárolnak állapotjellemzőket, státusokat (ezzel a vezérlést segítik).

A legtöbb regiszternek van neve - ezeket az assembly programozó használhatja.

Különböző hosszúságúak (bitszélességűek) lehetnek (1 bájtos, 2 bájtos szó stb.), ezeken belül lehetnek „átlapolások".

A regisztereket többféle módon osztályozhatjuk. A programozási felhasználási lehetőségek szerint vannak a *programozó számára látható* (user visible) regiszterek. Ezeket mind az alkalmazások, mind a rendszerprogramok használhatják. Ezen az osztályon belül a felhasználási mód szerint vannak

- *általános célú regiszterek*, melyeknek felhasználási módjuk nem kötött, melyek a gépi instrukciók argumentumaiban általánosan szerepelhetnek. Vannak ezen kívül
- *speciális célú regiszterek*, melyek korlátozottan használhatók. A korlátozás azt jelenti, hogy csak bizonyos instrukciók argumentumaiként szerepelhetnek.

A *programozó számára nem látható regisztereket* a processzor használja saját működésének kontrolljához.

A kimondottan a felhasználás célja szerint is osztályozhatunk. Ekkor vannak

- *adatregiszterek* (R0-Rxx, AX stb.***)***. Adatelemek tárolására szolgálnak. Az ALU a „kalkulációkat" (részben) az adatregisztereken tudja végrehajtani.
  - Jó, ha sok van belőlük.
  - Különböző hosszúságúak lehetnek, különböző adattárolási formátumuk lehet (8, 16, 32 stb. bites, fix- és lebegőpontos regiszterek).
- Vannak *címregiszterek* is. Adatok és instrukciók memóriabeli címeinek tárolására szolgálnak, a címzés segítik. Több alosztályuk lehet.
  - *Általános célú címregiszterek* azok, melyeknek általában a címzésekkel kapcsolatosan használhatók. Ilyen lehet az *indexregiszter* (az indexregiszteres címzéshez a bázis címhez adandó idex értéket tartalmazhatja), a *szegmensregiszterek* (a tartalmukhoz adandó eltolás érték adja a címet).
  - Az *utasításmutató regiszter* (PC: Program Counter, v. IP: Instruction Pointer), ami mindig a soron következő instrukció tárbeli címét tárolja.
    Saját hardver inkrementációja van, az instrukció feldolgozása során automatikusan növekszik a tartalma (nem szükséges gépi instrukcióval léptetni). Hallatlanul fontos a Neumann elvben! Az „ugrások" (jump, branch) implementációja pedig éppen a PC/IP megváltoztatásával előállítható, és lehetnek utasítások, melyeknek argumentuma éppen a PC/IP.
  - A *verem-mutató regiszter* (SP: Stack Pointer) szintén fontos címregiszter. Miután több szintű veremtár létezhet, több SP is lehet. A „veremkezelő instrukciók" (PUSH, POP) automatikusan hivatkoznak rá és automatikusan állítják.
  - Általában a programozó számára nem látható címregiszterek a virtuális címzéshez használandó *címleképzési táblákat mutató regiszterek.*

- Speciális célú regiszterek az
  - *állapotregiszter(ek)*. A processzor belső állapotát jellemző biteket tartalmaz.
    Ilyen bitek: C - átvitel (carry) bit,
    Z - zero bit,
    S - előjel (sign) bit,
    O - túlcsordulás (owerflow) bit,
    P - paritás bit,
    H - half carry bit stb.
    A jelzőbitek az instrukciók végrehajtása során bebillennek, vagy törlődnek: jeleznek egy-egy állapotot. A feltételes ugró instrukciók éppen a jelzőbiteket használják a feltételre: lesz tehát pl. *jump on Z bit* instrukció.
  - Egyéb *vezérlőregiszter(ek)* is lehetnek. Az üzemmód állapotot (lásd később: felhasználói mód - kernel mód) illetve a megszakítás (interrupt) maszkot tartalmazhatják.
    Sok processzorban ez a két (állapot és vezérlő) regiszter együtt a PSW (Program Status Word), a *program állapot leíró* szó. Néhol e két regiszter és a PC/IP (Program Counter/Instruction Pointer) együtt a PSLW (Program Status Longword), a *program állapot leíró hosszúszó*.

5.3. A vezérlő és dekódoló egység

Feladata a „felhozott” (fetched) gépi instrukció elemzése, dekódolása, és a CPU többi elemének, különösképpen a *végrehajtó egységnek* (ALU és regiszterek, esetleges *védelmi egységnek*) összehangolt működtetése.

5.4. A címképző és buszcsatoló egység

A *címképző egység* alapfeladata az ún. *virtuális címek* leképzése *valós címekre*. Ezt szoros együtműködésben végzi az *operációs rendszer* megfelelő komponenseivel, ha van a processzszorban *védelmi egység*, akkor ezzel is. A leképzésekkel az Operációs rendszerek c. tantárgy keretein belül foglalkozunk.

A *buszcsatoló egység* kezeli a sínt (síneket), adatforgalmat bonyolít le.

5.5. A CPU belső sínje, sínjei

Ez a processzoron belüli adatforgalmat biztosító áramkörök összessége. (A sínekről általánosan később még lesz szó.)

5.6. Az utasításkészlet

A gépi utasítások (Machine Instructions) (továbbiakban instrukciók) általános szerkezete:

Műveleti kód	Címrész

Az instrukciók címrésze határozza meg, mi az instrukció operandusa. Természetesen vannak két, esetleg lehetnek három operandusú instrukciók is, ezeknek rendre két illetve három cím-

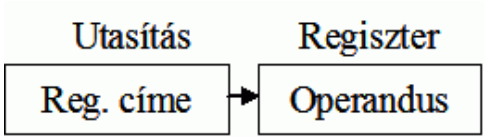
részük van. A CPU architektúra (első értelem!) specifikálja, milyen instrukciókat képes a CPU végrehajtani.

A fejlesztők szempontjai az instrukciókészlet kialakításához:

- Kódsűrűség növelése: adott instrukcióméret mellett minél több instrukció kódolható legyen.
- Ortogonalitás: bármely instrukció mellett bármely címzési mód lehetősége biztosított legyen.
- Szisztematikus kódolás: az instrukcióban lehetőleg egyforma mezők legyenek, melyeknek szerepe minden instrukcióban ugyanaz.
- Kompatibilitás meglévő rendszerekhez: architektúra családokhoz tartózó processzorok így alakulhatnak ki, korábbi programok is használhatók, de túlhajszolása hátráltatja a fejlődést.
- Operációs rendszerek, compiler-ek támogatása.
- Növekvő bitszám, ekkor
  - könnyebb az instrukciók szisztematikus kódolása,
  - gyorsabb az ALU működése,
  - nagyobb a címtartomány.
- Milyenek legyenek az adattípusok (bit, BCD, byte, szó, hosszú szó, lebegőpontos ábrázolások, szöveglánc stb.).

5.7. Címzési módok

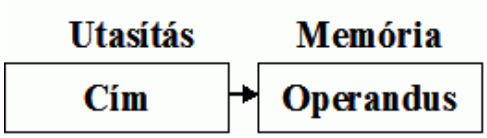
Az instrukciókban lévő címrész többféle címzési móddal határozhatja meg az instrukció operandusát. A „szokásos” címzési módok a direkt regiszter vagy direkt memória címzés, az indirekt memória címzés, az indirekt regiszter címzés (ennek normál és pre/post auto in/dekremens változataival), a bázisregiszteres címzés és a közvetlen címzés.



5.2. ábra. Direkt regiszter címzés

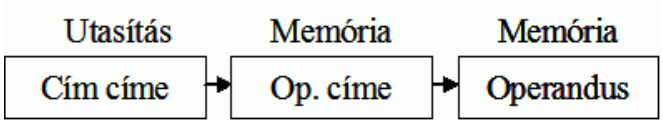
Direkt címzés

- *Direkt regiszter címzés*: címrészben regiszter címe található, a hivatkozott regiszterben pedig az operandus (5.2. ábra). Rövid operációkódot ereményez, egyszerű, gyors az istrukció elemzés, dekódolás.



5.3.ábra. Direkt rekesz címzés

- *Direkt rekesz címzés*: címrészben memória rekesz címe található. A memória cellában az operandus (5.3. ábra). Az instrukció hosszú, dekódolása egyszerű és természetes. Valós címzésű memóriamenedzselés esetén em relokálható a kód.

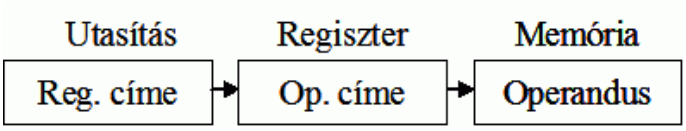


5.4. ábra. Indirekt memória címzés

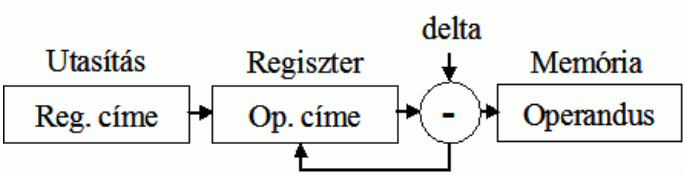
Indirekt címzés

- *Indirekt memória címzés*: címrészben memória rekesz címe, ebben rekeszben az operandus

címe van (5.4. ábra). Hosszú kód az eredmény, valós címzés esetén nem relokálható a kód. Összetett adatstruktúrák kezelésére jó ez a címzésfajta.

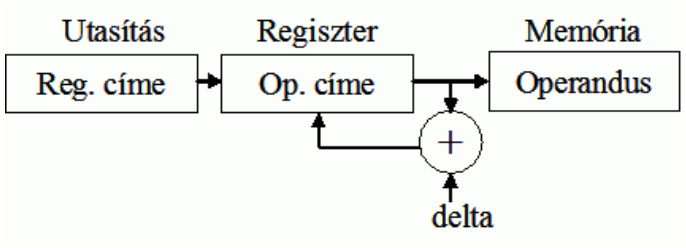


5.5. ábra. Indirekt regiszter címzés

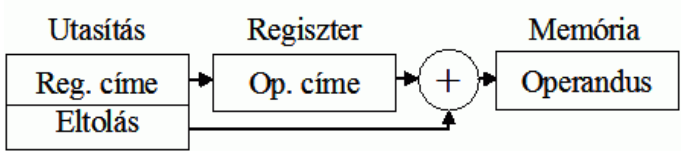


5.6.ábra. Pre-auto dekremens címzés

használat pop instrukciója ilyen (5.7. ábra).

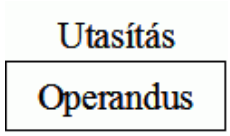


5.7.ábra. Post-auto inkremens címzés



5.8. ábra. Bázistregiszteres címzés

- *Közvetlen* (immediate) *címzés*: tulajdonképpen PC/IP relatív címzés eltolás nélkül, az operandus magában az instrukcióban van, vagyis az instrukció kód után közvetlenül (5.9. ábra).



5.9. ábra. Közvetlen címzés

- *Indirekt regiszter címzés*: a címrészben regiszterhivatkozás, a regiszterben az operandus címe. Néha maga az instrukció kód (push, pop) implicite hivatkozik a veremmutató regiszterre (SP-re). Alesetei:
  - - *normál* (5.5. ábra).
  - *pre-auto dekremens címzés*: tipikusan a veremhasználatnál a *push* instrukciónál (5.6.ábra).
  - *post-auto inkremens* címzés: tipikusan a verem-

Relatív (bázisregiszteres és közvetlen) címzés

- *Bázisregiszteres címzés*: az instrukcióban regisztercím és eltolás érték van kódolva. A regiszter tartalma egy memória rekesz cím, melyhez adva az eltolás értéket kapjuk az operandus címét (5.8. ábra). Hosszú a kódot eredményez, de a program relokálható (áthelyezhető), hiszen a bázisregiszter változtatásával más kiindulóponttól címezhetünk.

5.8. Instrukciókészletek, instrukciók csoportjai

Sokféle processzor van, különböző instrukciókészletekkel. Reménytelen lenne mindet felsorolni, megtanulni.

De vannak alapvető fontosságú instrukciócsoportok. Nézzük ezeket, úgy, hogy néhány példát is felsorolunk az egyes csoportokban. A példákban az instrukciók egy elképzelt assembly mnemonikjait adjuk meg, mert ez olvashatóbb, érthetőbb (tehát nem egy elképzelt gépi kódot).

1. Adatmozgató instrukciók

```
LOAD, STORE, LB, LW, SB, SW, ...
MOVE
IN, OUT
```

2. Aritmetikai és logikai instrukciók

```
ADD, SUB
MUL, DIV
AND, OR, XOR, NOT
NEG, COMPL csoport (Komplemensképző)
TEST
COMPARE csoport
```

3. Ugró instrukciók (Jump és Branch)

- feltétel nélküli

```
JUMP, BRANCH
```

- feltételes

```
J(feltétel): JZ, JS, ...
```

4. Bitléptetések, bitforgatás, inkrementáció, dekrmentáció, jelzőbeállítások

```
SHIFT, SLL, SRL, SRA, ...
RCL, RCR        (Rotate L/R carryn át)
ROL, ROR        (Rotate L/R)
INC, DEC
SET csoport
CLEAR csoport
```

5. Eljárás/függvényhívás, IT hívások, visszatérések instrukciói (CISC)

```
CALL, RET, (LEAVE)
IT, IRET
SYSCALL (RISC)
BREAK, HALT
WAIT
NOP
```

6. Ciklusszervező instrukciók (CISC)

```
LOOP
REP (Repeat stringműveletekre)
```

7. Veremkezelő instrukciók

```
PUSH, PUSHA, ...
POP, POPA, ...
```

8. Társprocesszor instrukciók

```
FINIT   (Társprocesszor inicializálás)
FLD             (Töltés a veremre)
FST             (Leemelés veremről)
FADD, FSUB, FMUL, .. (Aritmetikai instr.)
FWAIT   (Szinkronizációhoz)
```

5.9. Processzorok működési módjai

A korszerű operációs rendszerek működéséhez elvárjuk a korszerű processzoroktól, hogy legyen legalább két - egymástól jól megkülönböztethető - működési módjuk. Ezek szokásos nevei:

- normál mód (v. felhasználói mód),
- védett mód (v. kernel mód).

(Egyes processzoroknak több (egyre privilegizáltabb) üzemmódja is lehet.) Az üzemmódot - mint működési állapotot - a CPU nyilvántartja.

A védett v. kernel mód beállítása mély operációs rendszerbeli feladat, szokásosan ezt a "trap" (csapda) konstrukción keresztül végeztetjük.

Az üzemmódok közötti lényegbeli különbségek:

- védett (vagy több módnál egyre privilegizáltabb) módban szélesebb az instrukciókészlet: azaz bizonyos instrukciókat a CPU csak privilegizáltabb módban tud végrehajtani.
- védett (vagy privilegizáltabb) módban szélesebb címtartományt képes a CPU kezelni: azaz normál (kevésbé privilegizált) módban bizonyos címeket nem "lát" a processzor.

Az üzemmód váltás egyszerű felhasználói programokból nem lehetséges. Ezt csak az operációs rendszer magjának (kernel) hívásával, a trap konstrukción át érhetik el az alkalmazások.

A. függelék: A verem (stack) adattípus formális specifikációja

A formális specifikációhoz meg kell adni:

- a típusokat, amikből az új adattípus épül (konstruálódik);
- az operációkat (operátorokat) definíciószerűen;
- az axiómákat és
- a peremfeltételeket.

Emlékezzünk! Mire kellett egy compilernek a típus?

- Milyen a helyfoglalása és milyen az implementációja (ez együtt megadja az értékkészlete).
- Milyen operátorok vannak a típuson, és ezek hogy hatnak. (Ez a szintaktikus ellenőrzéshez is kellett:)

Típusok a verem adatszerkezethez:

```
Type    X:    akármilyen ismert típus;
        B:    boolean;
        S[X]: X-ek verme;
```

Funkciók:

```
new(x: X)               -> S[X]  ! létrehozás
empty(s: S[X])   S[X]   -> B     ! üresség vizsgálat
full(s:S[X})     S[X]   -> B     ! teliség vizsgálat
push(x: X)       x,S[X] -> S[X]  ! x-et verem tetejére
pop(s: S[X])     S[X]   -> X     ! tetejéről levétel
grow(s: S[X])    S[X]   -> S[X]  ! verem növelése
delete(s: S[X])  S[X]   ->       ! verem megszüntetése
```

Axiómák: `V(x: X, s: S[X])`

```
      empty(new(x))
  not empty(push(x))
      pop(push(x))
```

Peremfeltételek (preconditions) `V(x: X, s: S[X])`

```
 pre pop(s)     not empty ( s )
 pre push(x)    not full ( s )
 pre delete(s)  empty ( s )
 pre new(x)     van hely a heap-ban
 pre grow(s)    van hely a heap-ban
```