10. tétel második része:

„Az Intel x86 utasításkészlet-architektúrája (regiszterek, címzési módok, utasítások, memória architektúra, megszakítási rendszer)”

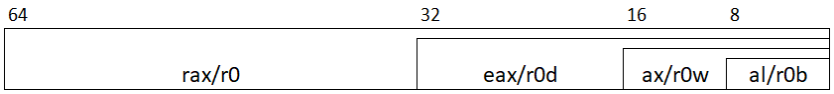
Az x86 jelölés a mikroprocesszorok utasításkészlet-architektúra (ISA) szerinti sorozatára vonatkozik. Ennek a sorozatnak néhány történelmileg fontos tagja: Intel 8086, Intel 80386, Pentium, AMD Athlon, Intel Core i3/i5/i7, AMD Ryzen 3/5/7. Ezek közül a 32 bites változatokat szokás IA-32 architektúrának is nevezni. Az 1978 óta tartó folyamatos fejlesztések során törekedtek a visszafelé kompatibilitásra. A processzorcsalád elemeit több gyártó is megvalósította és széles körben alkalmazták/alkalmazzák őket személyi számítógépekben. Az x86 architektúra fix pontos aritmetikát alkalmaz, de együttműködésre képes a lebegőpontos aritmetikát (és így valós számokat) használó x87 architektúrán alapuló társegységgel (FPU).

A CISC (összetett utasításkészlettel rendelkező számítógép) alapokra épülő rendszer megismeréséhez át kell tekintenünk az utasításkészlet-architektúra főbb elemeit, azaz a számítógép-architektúra programozáshoz kötődő részeit:

- regiszterkészlet
- utasítások
- címzési módok
- memóriakezelés
- megszakítási rendszer

Regiszterkészlet

A programozó különböző méretű regisztereket használhat, azonban a kisebb regiszterek nevei mindig a nagyobb regiszterek részeinek álnevei. A regiszternevek utalnak a méretre. A ’h’ és’l’ betűkre végződő regiszterek 8 bitesek, akétkarakteres ’x’ karakterre végződő regiszterek 16 bitesek, az ’e’ betűvel kezdődő három karakteres regiszternevek pedig 32 bitesek. (Később a 64 bites x86-64 kiterjesztésben megjelennek az ’r’ karakterrel kezdődő 8 bájtos regiszterek is, bár ezek kisebb részeire máshogy is lehet hivatkozni.)



A programozó 8 általános célú regisztert használhat: `eax`, `ebx`, `ecx`, `edx`, `esp`, `ebp`, `edi`, `esi`. Ezek azonban nem teljesen egyenértékűek, egyes regiszterek speciális jelentéssel/tulajdonsággal bírnak. Például az `esp` regiszter (veremmutató) a rendszerverem legfelső elemének címét tárolja. Az `ebp` regiszter (bázis pointer) dinamikus élettartamkezelésű lokális változók címzésekor használandó. Az `eax` regiszter az osztás és szorzás műveleténél tölt be speciális szerepet, valamint a egész értékkel visszatérő függvények is itt adják vissza a visszatérési értéket. Az `edx` regiszternek is fontos szerepe van az osztás és szorzás során, valamint az I/O műveleteknél. Az `ecx` regiszter használható ciklusszámlálóként. Az `edi` és `esi` regiszterek indexként használhatóak sztring műveletek során, valamint a

paraméterátadásban is fontos szerepük van. Az x86-64 esetén további 8 darab 64 bites általános célú regiszter (`r8-r15`) is rendelkezésünkre áll.

Vannak még további speciális rendszer regiszterek. Az `eip (rip)` regiszter (instruction pointer, program számláló) a következő gépi utasítás memóriacímét tartalmazza, ezáltal a szekvenciális végrehajtásban és a vezérlésátadás során nélkülözhetetlen. A `eflags (rflags)` regiszter tölti be az állapot/státusz regiszter szerepét. Ennek bitjei egy állapot meglétét vagy hiányát jelzi. Többek között beszélhetünk átvitel bit (`CF`), túlcsordulás bit (`OF`), előjel bit (`SF`), zéró bit (`ZF`), megszakítási maszk bit (`IF`) stb. A különböző memóriaszegmensek használatához szükséges szegmens szelektor regisztereket is használhatunk (`CS, DS, SS, ES`).

Használhatóak még lebegőpontos x87 regiszterek és a további kiterjesztések (MMX, SSE, AVX, AVX-512) által nyújtott regiszterek is.

Utasítások

CISC rendszer révén sok összetett utasítás érhető el. Az utasítások elérhetik a memóriát, azaz az operandus legtöbb esetben lehet konstans, regiszter és memóriában tárolt érték is. Többnyire kétcímes utasításokat használhatunk és az utasításokhossza változó. Az utasítások az operandus méretét is meghatározzák. A fontosabb utasítás kategóriák:

- Adatmozgató utasítások (`mov, push, pop, lea, movsx, xchg, cdq`, …)
- Aritmetikai és logikai utasítások (`add, sub, imul, idiv, inc, dec, neg, cmp, and, or, xor, not, shl, shr, sar, ror, rcr`, …)
- Vezérlésátadó utasítások (`jmp, je, jne, jg, jge, jl, jle, jb, ja, jz, jnz, jc, jo, call, ret, leave, int, loop`, …)
- Egyéb utasítások (`nop, cpuid, sti, cli`, …)

Címzési módok

Az utasítások operandusainak „elérési módjait” határozhatjuk meg a különböző címzési módok segítségével. Nem megengedett egy utasításon belül mind a cél, mind a forrás operandus esetén memóriahivatkozás megadása. A főbb címzési módok a következők:

- Közvetlen adat (immediate): Az operandus egy konstans, amely a gépi kódú utasítás részét képezi a kódszegmensben.
- Regiszteres címzés: Az operandus a megadott regiszterben található.
- Közvetlen címzés (direct): Az operandus egy memóriacím, ahonnan/ahova a szükséges érték beolvasható/kiírható.
- Közvetett regiszteres címzés (register indirect): Egy megadott regiszter tartalmazza a felhasználandó érték memóriacímét.
- Bázis-relatív címzés (Base/register relative): Egy bázisként szolgáló regiszterben lévő memóriacímhez képest egy bizonyos eltolási értékkel arrébb lévő memóriaterület használata.

- Indexelt címzés: Egy adott memóriacímhez hozzá kell adni egy indexként használt regiszterben lévő értéket az operandus megtalálásához. Az index értékét egy konstans értékkel meg is lehet szorozni, mielőtt hozzáadjuk a kiindulási címhez (skálázás).

Ezeknek a címzési módok kombinált használatának lehetőségeit foglalja össze az alábbi kifejezés:

```
[<szegmens>:] [<bázis>] [+<index> [*<skála>]] [+<eltolás>]
```

Itt a `<szegmens>` a szegmens szelektor regisztereket jelenti, a `<bázis>` és az `<index>` általános célú regiszterek lehetnek, a `<skála>` és az `<eltolás>` pedig konstansok, előbbi az 1,2,4 vagy 8 értékeket veheti fel. (A szögletes zárójel opciót jelent.)

Memória architektúra

A korai rendszerek technikai korlátjai miatt volt szükség a szegmentált memória használatára. Később kompatibilitási okok miatt sokáig fennmaradt. Lényege, hogy a 8086 processzor által használt 2 darab 16 bites regiszter segítségével lehessen előállítani 20 bites fizikai címeket. Egy szegmens szelektor regiszter értékének 16 szorosát (4 bites eltolás) kell hozzáadni egy általános regiszter értékéhez és így áll elő a szükséges operandus címe. Így tehát egy szegmens 64kB méretű, a teljes címtér pedig 1MB volt.

Ez használta az eredeti működési mód, amit a címtér kiterjesztését és bizonyos védelmi funkciókat megvalósító új *védett* mód bevezetése után elneveztek *valós* módnak. A 32 bites rendszerekben használhatunk még ún. *virtuális* működési módot, a 64 bites rendszerekben pedig a *hosszú* módot.

A regiszterkészlet támogatja a különböző szegmensek használatát. A kód szegmensben helyezkedik el a program gépi kódja. Szegmens regisztere a `CS` (ami például az `EIP`-vel „működik együtt”). Az adat szegmensben találhatóak a statikus élettartamkezelésű változók, ennek eléréséhez a `DS` regiszter szükséges. A rendszerverem a stack szegmensben helyezkedik el, alapértelmezetten az `SS` szegmensregiszterrel dolgozik együtt az `ESP` és az `EBP` regiszter.

Megszakítási rendszer

Háromféle megszakítást kezelhetünk:

- Hardveres (külső eszközök a megszakítási igényüket így jelezhetik, IRQ)
- Szoftveres (a program kér megszakítást az `int` utasítással)
- Kivételek (a processzor által generált megszakítás, pl. division-by-zero, page fault)

A prioritással is rendelkező megszakítási kérelmeket egy PIC (programozható megszakítás vezérlő, pl. Intel 8259A) dolgozza fel és továbbítja a processzornak. A processzorban a megszakításkezelés maszkolható az `EFLAGS` regiszter `IF` bitje révén. Vektoros megszakítást alkalmazva a megszakítást kérőhöz tartozó IDT (megszakítás leíró tábla) bejegyzés alapján találjuk meg a megfelelő ISR (megszakításkezelő rutin) címét. Megszakításkezelése előtt természetesen kontextus mentésre van szükség (azaz többek között az `EIP`, a `CS`, az `EFLAGS` regiszterek tartalma a verembe lesz mentve).

Megjegyzés:

A tételhez kapcsolódó hardveres és assembly programozói ismeretek szükségesek lehetnek. A témakörrel kapcsolatos tágabb ismeretség nem hátrány.

Kapcsolódó tantárgyak:

- *Számítógép architektúrák*
- *Assembly programozás*

Ajánlott irodalom:

- Joseph Cavanagh: X86 Assembly Languages and C fundamentals (CRC Press, 2013)
  2. és 3. fejezet.
- Richard Blum: *Professional Assembly Language* (Wiley, 2005)
  2. fejezet.