7. tétel második része:

„Vezérlési szerkezetek implementálása assembly nyelven (feltételes és feltétel nélküli vezérlésátadás, elágaztatás, ciklus szervezés, alprogramhívás)”

A processzorok működésük során a memóriába betöltött gépi kódú program utasításait hajtják végre. A CPU programszámláló regisztere (PC, EIP) a programvégrehajtás során mindig a következőként végrehajtandó utasítás memóriacímét tartalmazza. Az elemi utasításvégrehajtás (fetch-execute ciklus) fontos eszköze ez a regiszter a vezérlőegységben. Az esetek többségében egy utasítás végrehajtása során a programszámláló regiszter értéke az adott utasítás hosszával nő (szekvenciális végrehajtás).

Vannak azonban olyan esetek (elágazás, ciklus, alprogram hívás, stb.) amikor az aktuális utasítás után nem a memóriában közvetlenül mögötte lévő utasítást kell végrehajtani, hanem egy olyan utasítást, amely a RAM-ban valahol máshol van. Ilyenkor a programszámláló nem egyszerűen csak „inkrementálódik”, hanem a programban megadott másik memóriacímmel íródik felül. Ezt vezérlésátadó utasításokkal teheti meg a programozó, ezáltal különböző vezérlési szerkezeteket létrehozva.

Assembly programozás szintjén feltételes és feltétel nélküli utasítások révén tudjuk közvetlenül megváltoztatni a programszámláló értékét és a magasszintű programozási nyelvek vezérlésátadó utasításai (például: `if-else`, `switch`, `while`, `for`, `break`, `continue`, `goto`, `return`) is ezekre képződnek le fordítás/interpretálás során. (*A továbbiakban bemutatásra kerülő assembly utasítások Intel szintaxisú x86 utasításkészlet architektúrán alapulnak.*)

Feltétel nélküli vezérlésátadás

Ebben az esetben az adott utasítás végrehajtása során a programszámláló minden esetben egy megadott értékkel íródik felül. Ez lehet egy operandusként a programban definiált érték vagy például korábban a rendszerverembe helyeztett érték. A fontosabb utasítások a következők:

- `jmp`
  Az operandusként megadott cím kerül minden esetben a programszámlálóba. Természetesen az operandus assemblyben lehet, hogy címkeként van megadva, de a gépi kódban már (abszolút vagy relatív) memóriacímre utal. Ez az utasítás húzódik meg például a C nyelv `goto`, `break` és `continue` utasítása mögött.
- `call`
  Az alprogramhívás utasítása. Az operandusa a meghívni kívánt eljárás vagy függvény első utasításának memóriacíme, így kerül át a vezérlés a hívott programegységhez. Mielőtt felülíródik a programszámláló ezzel a címmel az aktuális értéke, mint későbbi visszatérési cím a verem tetejére mentődik.

- `ret`
  Alprogramból való visszatérésre szolgál. Végrehajtása során a verem tetején lévő érték (a híváskor elmentett visszatérési cím) kerül a programszámláló regiszterbe, azaz a végrehajtás szempontjából következő utasítás a memóriában a legutóbbi hívó utasítás után elhelyezkedő utasítás lesz. Ezt alkalmazza a C fordító a `return` utasítás implementálása során.
- Ebbe a kategóriába sorolható még például a `leave`, az `int` vagy az `iret` assembly utasítás is.

Feltételes vezérlésátadás

Az adott ugró utasítás végrehajtása során dől el, hogy hogyan változik a programszámláló regiszter tartalma. Ezt a processzor állapota, mint egyfajta feltétel határozza meg, azért a végrehajtás során az állapotregiszter (SR, FLAGS) nélkülözhetetlen szerepet tölt be. Vagy csak hagyományos értelemben „inkrementálódik" a programszámláló regiszter tartalma (átáll a következő memóriacímen található utasításra) vagy egy operandusként megadott memóriacímmel íródik felül (azaz valahol máshol folytatódik a program végrehajtása).

- `jz/jc/jo/js`
  Az utaítás mnemonikjának második betűje határozza meg a feltételt, ami a státusz regiszter adott bitjének (rendre ZF, CF, OF, SF) pillanatnyi értékétől függ. Ha az adott flag bit be van állítva (azaz értéke 1), akkor az operandusként megadott cím kerül a programszámlálóba (ugrás). Ellenkező esetben az operandus nem kerül felhasználásra és a programszámláló regiszter tartalma a feltételes ugró utasítás hosszával nő.
- `jnz/jnc/jno/jns`
  Ezek az utasítások hasonlóan működnek, mint az előzőek, annyi eltéréssel, hogy a feltételek tagadva vannak, azaz a konkrét flag bit 0 értéke eredményez ugrást.

Léteznek olyan feltételes ugró utasítások is, amelyek az előbbi (állapotregiszter bit alapú) utasítások kombinációi vagy egyszerű alias nevei. A `cmp` összehasonlító assembly utasítás a státuszregiszter zéró (ZF), átvitel (CF), túlcsordulás (OF) és előjel (SF) bitjét változtatja meg az operandus értékek egymáshoz való viszonya alapján. Így a könnyebb kódolvashatóság kedvéért használhatjuk a következő assembly mnemonikokat is:

- `je/jne`
  Az egyenlőség, illetve az eltérőség esetén szükséges ugrás leírása (a `jz/jnz` helyett).
- `jl/jle/jg/jge`
  Előjeles értékek esetén a $<$, $\leq$, $>$, $\geq$ esetek kezelésére.
- `jb/jbe/ja/jae`
  Előjel nélküli értékek esetén a $<$, $\leq$, $>$, $\geq$ esetek kezelésére.

Gyakran a fenti feltételes utasításokra együttesen a `jcc` jelöléssel hivatkozunk.

Speciális feltételes ugró utasítások a `loop`, `loope` és `loopne`, amelyek egyfajta ciklusszervezésre szolgálnak, ahol a vezérlésátadás megvalósulása az ECX általános célú regiszter aktuális értékétől (is) függ.

Elágaztatás

Magasszintű programozási nyelvekben többféle elágaztató szerkezetet használhatunk (például C-ben az `if`, az `if-else` és a `switch` szerkezeteket). Assembly szinten ezeket a fentebb tárgyalt feltételes és feltétel nélküli ugró utasításokkal kell megvalósítani.

Egy else ág nélküli if szerkezet „csináld meg vagy hagyd ki” logikáját egyetlen feltétel nélküli vezérlésátadó utasítás felhasználásával megvalósíthatjuk. A feltétel kiértékelést megvalósító utasítások után el kell helyeznünk egy feltételes ugró utasítást. Ezt követik azok az utasítások, amelyeket opcionálisan kell végrehajtani (azaz az if ág utasításai). Végül az összes többi (minden esetben szükséges) utasítás, de ezek közül az első címkével kell rendelkezzen, amit a feltételes ugró utasítás operandusaként is meg kell adni. A `jcc` utasításban a feltétel tagadását kell megadni, mivel azt határozzuk meg mikor kell átugrani az opcionális blokkot szemben a magasszintű nyelvekkel, ahol azt adjuk meg mikor kell végrehajtani azt.

A teljes if-else szerkezet esetén assemblyben először leírjuk a feltétel kiértékelés lépéseit, majd egy megfelelően kiválasztott `jcc` utasítást, aztán pedig az „igaz” blokkot. A „hamis” blokk utasításai csak ez után lesznek megadva. Utóbbinak az elejére el kell helyezni egy címkét, ami a `jcc` utasítás operandusa lesz. A két ág utasításai közé (vagyis az „igaz” ág végére) el kell helyezni egy feltétel nélküli ugró utasítást (`jmp`), aminek az operandusa a „hamis” ág utáni első utasítás címkéje lesz. Ezáltal elkerüljük, hogy a vezérlés az „igaz” ág végrehajtása után átfolyjon a „hamis” ágra. Tehát egy `jcc` és egy `jmp` utasítás, valamint két címke megadása szükséges.

Ciklusszervezés

Csak a feltételes ciklusokat mutatnám be. A C nyelvben az `for`, az `while` kezdőfeltételes ciklusok (assemblyben egyáltalán nincs köztök különbség), míg a `do-while` szerkezeteket végfeltételes ciklust takar.

Optimalizálási megfontolások miatt a kezdőfeltételes assembly ciklus programozása során a törzs utasításait írjuk le előbb és csak ez után jönnek a feltétel kiértékelésének lépései, valamint egy `jcc` utasítás. Ennek a feltételes vezérlésátadó utasításnak az operandusa a törzs első utasításának címkéje lesz. Az egész ciklusleíró szerkezet legelejére egy feltétel nélküli ugró utasítás kerül, aminek az operandusa a feltétel kiértékelés első utasításának címkéje kell legyen. Így mindenképpen a feltétellel kezdődik a végrehajtás és akár üres ciklust is kaphatunk.

A végfeltételes ciklus assembly kódja mindössze annyiban különbözik a kezdőfeltételesétől, hogy nem szerepel benne a `jmp` utasítás, így a törzs utasításai legalább egyszer lefutnak.

Alprogram hívás

Az alprogram hívó `call` utasítás és a hívásból való visszatérésre szolgáló `ret` utasítás már bemutatásra került korábban. Szorosan nem tartozik a szűk értelemben vett vezérlésátadáshoz a paraméterátadás és a visszatérési érték kezelése, így csak röviden kerülnek megemlítésre. A paraméterek átadására használhatunk regisztereket (`edi`, `esi`, `edx`, `ecx`) illetve egyes esetekben a rendszervermet is. A függvény visszatérési értékének

hívóhoz való visszajuttatásához általában egy regisztert (`eax`) használunk. Közben vigyáznunk kell a regiszterek és a rendszerverem konzisztenciájára.

Megjegyzés:

A témakörrel kapcsolatos tágabb ismeretség és assembly programozási jártasság előny.

Kapcsolódó tantárgyak:

- *Assembly programozás*
- *Számítógép architektúrák*

Ajánlott irodalom:

- R. E. Bryant, D. R. O'Hallaron: Computer Systems – A programmer's perspective (Pearson, 2016)
  3. fejezet.
- Joseph Cavanagh: X86 Assembly Language and C Fundamentals, (CRC Press, 2013)
  6. fejezet.
- Richard Blum: Professional Assembly Language, (Wiley, 2005)
  6. fejezet.