Assembly nyelvek (vázlat az előadásokhoz)

07/11/05 16:40:11

- Bevezetés: Irodalom, HLA, gépi kód, CPU, regiszterek, alapvető utasítások
- Memória elérés: címzésmódok, mutatók, memória szervezés
- Adatábrázolás: típusok, gépi kód, számrendszerek, előjeles egészek
- programszerkezeti elemek: if, ciklusok, eljárások, paraméter átadási módok, lokális változók
- Alacsony szintű kontroll szerkezetek
  - JMP oda; Jcc címke; LOOP(címke);
  - call elj; ret(n);
- Utasításkészlet:
  - Adatmozgatás és veremműveletek
  - Aritmetika és BCD korrekciók
  - Sztingműveletek, HLA-sztring szerkezete
  - Logikai és léptető műveletek
  - Megszakítások, kivételek, I/O, (LOCK, ESC, WAIT) (FPU, MMX)

Assembly nyelvek (vázlat az első ea.-hoz)

05/10/05 13:02:50

- A High Level Assembly (HLA) és más nyelvek – assembly-k és HLL-ek – gépi kód
- A HLA „Hellow World!” programja
  - szabad szintaxis
  - sztringek automatikus összefűzése
  - szerkesztés, fordítás, futtatás: **hla hw; hw**
- A 80x86 CPU család – Neumann gépek (?8.o)
  - CPU, Memória, I/O eszközök
  - cím busz, adat busz, kontroll busz
- CPU / Általános regiszetrek (8, 16, 32 bit)
- Alapvető utasítások: MOV, ADD, SUB
- Alapvető könyvtári függvények, adat deklarációk
  - #include(”stdlib.hhf”)
  - stdout.put
  - stdin.get
  - static, int32
  - Azonosítók (aláhúzás, betűk, számok)

„Hello World!” program

```
Program HelloWorld;
#include(”stdlib.hhf”)
begin HelloWorld;
stdout.put(”Hello World!”,nl);
end HelloWorld;

Program hw; // Sok-sok megjegyzés
#incule(”stdlib.hhf”)
begin hw;
stdout.put  // „szabad” szintaxis
(
   ”Hello”  // stringek automatikus összefűzése
     ” World!” nl  // nl == new line
);
end hw;
// Fordítás, futtatás: hla hw; hw
```

Neumann architektúrájú számítógépek

(Az Intel 80x86 CPU családja is ebbe a körbe tartozik)

- Fő részei:
  - CPU
  - Memória
  - I/O eszközök
- Kapcsolat közöttük: Rendszerbuszok
  - Cím busz (32 bit)
  - Adat busz (32 bit)
  - Kontroll busz

Az Intel 80x86 CPU regiszterei

- Általános célú regiszterek
- Speciális regiszterek
  - alkalmazások számára hozzáférhető
    - Flag regiszter (F, EF)
    - (Utasítás számláló (IP, EIP))
  - operációs rendszer számára fenntartott
- szegmens regiszterek
  - CS: Kód
  - SS: Verem
  - DS: Adat
  - ES, FS, GS: Extra adat

Az Intel 80x86 CPU általános regiszterei

- 8 bites regiszeterek
  - AL, AH, BL, BH, CL, CH, DL, DH
- 16 bites regiszterek
  - AX, BX, CX, DX, SI, DI, BP, SP
- 32 bites regiszterek
  - EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
- Átfedés van közöttük:
  - AX=EAX%65535=256*AH+AL , BX, CX, DX
  - SI=ESI%65535 , DI, BP, SP
- „jelentés” a név mögött:
  - AX=**A**ccumulator, BX=**B**ase register,
    CX=**C**ounter register, DX=**D**ata register,
    SI=**S**ource **I**ndex, DI=**D**estination **I**ndex
    BP=**B**ase **P**ointer, SP=**S**tack **P**ointer (NE HASZNÁLD!)
  - AL=**A**ccumulator **L**ow, AH=**A**ccumulator **H**igh, stb...
  - EAX=**E**xtended **A**ccumulator, stb...

A flag regiszter

Egyedi „zászlók” - logikai változók – gyűjteménye

- Állapot jelzők
  - bit-0: CF – **C**arry – átvitel
  - bit-2: PF – **P**arity – párosság
  - bit-4: AF - **A**uxiliary carry – közbenső átvitel
  - bit-6: ZF – **Z**ero – zéró
  - bit-7: SF – **S**ign – előjel
  - bit-B: OF – **O**verflow – Túlcsordulás
- Vezérlő bitek
  - bit-8: TF – **T**race – nyomkövető
  - bit-9: IF – **I**nterrupt enable – Megszakítás engedélyezés
  - bit-A: DF – **D**irection – Irány jelző

----ODIT.SZ-A-P-C

Alapvető utasítások

MOV(*mit*, *hová*);*// hová=mit*

- *mit*: regiszter, memória, konstans
- *hová*: regiszter, memória
- *mit* és *hová* azonos méretű (8, 16 vagy 32 bit)
- Flag regiszter változatlan

ADD(*mit*, *mihez*);*// mihez=mihez+mit*

- *mit*: regiszter, memória, konstans
- *mihez*: regiszter, memória
- *mit* és *mihez* azonos méretű (8, 16 vagy 32 bit)
- Állapotjelzők állítódnak

SUB(*mit*, *miből*);*// miből=miből-mit*

- *mit*: regiszter, memória, konstans
- *miből*: regiszter, memória
- *mit* és *miből* azonos méretű (8, 16 vagy 32 bit)
- Állapotjelzők állítódnak

INTMUL(forrás,cél); INTMUL(konstans,forrás,cél)

- *forrás*: regiszter, memória, konstans
- *cél*: regiszter
- *forrás* és *cél* azonos méretű (16 vagy 32 bit)
- Állapotjelzők állítódnak (CF=OF; többi határozatlan)

Alapvető könyvtári függvények, adat deklarációk

```
program plST;  // Azonosítók: [_a-zA-Z][_a-zA-Z0-9]*
#include(”stdlib.hhf);
static a:int32:=-7;
       b:int32;
begin plST;
stdout.put(”a értéke: ”,a,nl); // Írás a „képernyőre”
stdout.put(”írj be egy számot:”);
stdin.get(b);          // Olvasás a „billenytűzetről”
stdout.put(”A beírt szám: ”,b,nl);
end plST;

//„static”   : adat deklarációs rész megkezdése
//„a:”, „b:” : Változók nevének megadása („neutrális”)
//„int32”    : típus - 32 bites előjeles egész szám
//„:=-7”     : kezdőérték megadása  -- nem kötelező
```

Egyszerű minta program

```
program Demo;
#include(”stdlib.hhf”);
STATIC
   a:int32:=5;
   b:int32;
   c:int32;
begin Demo;
stdout.put(”Az 'a' változó értéke: ”,a,nl);
stdout.put(”Irj be egy számot:”);
stdin.get(b);
mov(b,eax);add(a,eax);if(@S) then neg(eax); endif;
mov(eax,c);
stdout.put(a,”+”,b,” absolut értéke: ”,c,nl);
end Demo;
```

Általános memória használat

```
......: storage szekció   ---
......: static szekció    ----(@nostorage)
......: read-only szekció --- (align(4)  )
......: rendszer konstansok
......: program kód       --- byte 3,7,9;(pseudo opcode)
16  MB: halom (default méret)
16  MB: verem (default méret) (VAR szekció)
128 KB: Operációs rendszer

Alignment:1-2-4-8-16, „nagy objektumok”: 8-16
align(x);
```

Adatábrázolás, adattípusok

- bit, nible, byte
- Assembly alaptípusok
  - byte, word, dword, qword, tbyte, lword

---------- valahol itt kezdődik a harmadik e.a.----

- memória elérés (align)
- Utasítások - címzésmódok
- „Magas szintű” adattípusok
  - Egész számok (int32, uns8)
    - bináris <=> BCD (Binary Coded Decimal)
    - pozitív <=> előjeles
  - Valós számok (real32, real64, real80)
  - Karakterek, karakterfüzérek (char, string)
  - Mutatók - memória címek (pointer)
  - Tömbök (tomb:int16[3,2];tomb2:byte:=[5,8,4,9];)

A gépi kód (ADD utasítás egyik formája)

- %000000_d_w %mm_reg_r/m [DISP vagy disp]
  %000000 -> ADD ; d -> melyik a forrás ; w -> méret
  mm -> mod (0-1-2: memória, 3 regiszter)
  regiszter:
  8 bit: AL,CL,DL,BL,AH,CH,DH,CH
  16 bit: AX,CX,DX,BX,SP,BP,SI,DI
  r/m, ha memória: DISP+
  BX+SI,BX+DI,BP+SI,BP+DI,SI,DI,BP,BX
  Kivétel: Ha mm=0 és r/m=6 ==> (BP helyett) DISP
  (Van még konstans hozzáadása, ill. speciális esetek.)
- %100000_s_w %mm_000_r/m [DISP vagy disp] data (data)
  s -> adat előjelkiterjesztése
- %0000010_w data (data)
  ADD(kons,AL); ADD(kons,AX);
- SIB byte (Skála Index Bázis)
  mod: 00-nincs 01-8bites 02-32bites eltolas 03-regiszter
  mod<11 és r/m=4--> SIB bájt: %sk_ind_baz
  mod=00 és r/m=5 16-bites eltolas
  REGS: eax, ecx, edx, ebx, esp, ebp, esi, edi
- Prefixek: utasítás, cím méret, operandus méret, szegmens

Az EIP regiszter

Címzésmódok (operandusok megadási módjai)

1: konstans, azonnali
2: regiszter
  – Memória címzések:
3: direkt, közvetlen
4: regiszter indirekt
5: bázis relatív
5: indexelt
7: bázisrelatív indexelt

„Magas szintű” adattípusok

- Pozitív egészek (uns8, uns16, uns32, uns64, uns128)
  0 … 2^(n)-1
- Előjeles egészek ábrázolása (LSB és MSB)
  - előjelbit (80 bites BCD számok)
    -2^(n-1)+1 … +2^(n-1)-1
  - eltolással EE=PE-konst. (valós számok karakterisztikája)
    -konst. … 2^(n)-1-konst.
  - egyes komplemens
    -2^(n-1)+1 … +2^(n-1)-1
  - kettes komplemens (int8, int16, int32, int64, int128)
    -2^(n-1) … +2^(n-1)-1 int=uns-MSB*2^n
- Valós számok (real32, real64, real80)

  érték=(-1)^(S)*2^(karakterisztika)*(1+mantissza) (IEEE szabvány)

  real32: 1+ 8+23 10^38 6-7 jegy

  real64: 1+11+52 10^308 15-16 jegy

  real80: 1+15+64 10^4800 19-20 jegy
- Mutatók (pointer)
- Karakterek, karakter füzérek (char, string)
- Összetett típusok (tömbök, recordok, uniók)

Adatmozgató utasítások (Minden flag változatlan)

- MOV(forrás,cél); //Nincs (m,m), (sr,sr), (k,sr)
- MOVSX(forrás,cél_reg);MOVZX(forrás,cél_reg)
- XCHG(m/r,m/r); // azonos méret, NINCS m,m
- XLAT; // MOV([EBX+AL],AL)
- LEA(mem,cél_reg32);// “=” MOV(&mem,cél_reg32)
- LAHF; SAHF; // Load, Save az AH szempontjából ..
- LDS(mem48,reg32);LES(mem48,reg32);

VEREM műveletek

- PUSH(forrás);//mem, reg, 16/32 bit
  - PUSHW(forrás); // mem16, reg16, konst ESP-=2
  - PUSHD(forrás); // mem32, reg32, konst ESP-=4
- POP(forrás); //mem, reg, 16/32 bit
- PUSHF; PUSHFD;
- POPF; POPFD;
- PUSHA; PUSHAD;
- POPA; POPAD;

Aritmetikai utasítások

- ADD(forrás,cél);SUB(forrás,cél);
- ADC(forrás,cél);SBB(forrás,cél);
- INC(m/r); DEC(m/r);NEG(m/r);
- CMP(cél,forrás);

Feltételes ugrások CMP után

- J... címke;
  - pozitív mennyiségek hasonlítása után
    JA, JB, JE, JAE, JBE,
    JNA, JNB, JNE, JNAE, JNBE
  - előjeles mennyiségek hasonlítása után
    JG, JL, JE, JGE, JLE,
    JNG, JNL, JNE, JNGE, JNLE
- INTMUL(forrás,cél);intmul(kons,forrás,cél);
- MUL(forrás);IMUL(forrás);
- DIV(osztó);IDIV(osztó);//kivétel

Programszerkezeti elemek

- Elágazások
  - if(feltétel) utasítások {{elseif utasítások} else utasítások} endif
    - felt: @z, @ae, a<ebx, al<=6, <>, !=, =, ==, {!}boolean_var, reg {not} in low..Hi
- Ciklusok
  - while(felt) do ut... endwhile
  - for(ut;felt;utasítás) do ut... endfor
  - repeat ut... until(felt)
  - forever ut.... endfor
    - break; breakif(felt);
    - continue; continueif(felt);
  - begin nev; ut...; end nev;
    - exit nev; exitif(felt) nev;

logikai utasítások

- NOT(cél); // Minden flag változatlan
- OR(forrás, cél);
- AND(forrás,cél);
- XOR(forrás,cél);
- TEST(forrás,cél); // TEST(al,al); TEST(8,AH);

- forrás: mem/reg/konst
- cél: mem/reg
- cél és forrás
  - azonos méretű / 8-16-32 bit
  - nem lehet mindkettő memória cím

- OF=CF=0
- AF – határozatlan
- PF, ZF, SF értelemszerűen

Forgatások, léptetések

- SHL(sz,cél);//SHR, SAR
- ROL(sz,cél);//ROR, RCL, RCR
- SHLD(sz,forrás,cél);// SHL(sz,cél:forrás);
- SHRD(sz,reg,mem/reg);// SHR(sz,forrás:cél);

- Csak a **cél** változik meg
- sz: konstans vagy CL
- cél és forrás azonos méretű
- forrás csak regiszter lehet, csak 16-32 bites lehet
- Csak CF és OF változik meg a forgatásoknál
- AF határozatlan a léptetéseknél

Közvetlen bitelérések

- Flag regiszter bitjei
  - CLC, CLD, CLI, STC, STD, STI, CMC
- BT(hányadik, miben); // Bit Test CF-et állítja
- BTC, BTR, BTS; // BT and Complement, Reset, Set
- miben: mem/reg16/reg32
- hányadik reg/konst: 0--15/31
  - regiszter méret
  - 255 – ha hányadik konstans
  - limit nélkül, ha hányadik reg és miben mem
  - ha reg, akkor azonos méretű mibennel

- BSF(miben,hova);// legalacsonyabb helyiértékű 1-es bit sorszáma mibenben --> hova / **ha ZF=0**
- BSR(miben,hova);// legmagasabb helyiértékű 1-es bit sorszáma mibenben --> hova
  - miben és hova azonos méretű (16/32 bit)
  - hova: regiszter

Eljárások, paraméter átadás

- Procedure elj_nev(formális_paraméter_lista);@opciók;
  lokális deklarációk
  begin elj_nev;
  utasítások
  end elj_nev;
- Eljárás Hívása: nev(aktuális_paraméter_lista);
- Paraméter átadás
  - módja
    - VAL, VAR, RESULT, VALRES, ...
  - helye
    - verem, regiszter
- Regiszterek megőrése
- lokális deklarációk
  - láthatóság
  - élettartam
- exit elj_nev;exitif(felt) elj_nev;

CPU Állapot (regiszterek)megőrzése

```
Program nem_mukodik;
#include(“stdlib.hhf”)
procedure tiz_szokoz;begin tiz_szokoz;mov(10,ecx);
repeat stdout.put(' ');dec(ecx);until(@z);
end tiz_szokoz;
begin nem_mukodik;
mov(20,ecx);
repeat tiz_szokoz();stdout.put(“*”,nl);dec(ecx);
until(@z);
end nem_mukodik;
// Javítás:
// valaki (hívó/hívott) menti ECX-et
```

Lokális változók

```
program demo;#include(„stdlib.hhf”)
static i:uns32:=10;j:uns32:=20;
Procedure elso;var i:int32;j:uns32;begin eslo;
mov(10,j);for(mov(0,i);i<10;inc(i))do
stdout.put(„i,j=”,i,” „,j,nl);dec(j);endfor;
end elso;
procedure masodik;var i:int32;begin masodik;
mov(10,j);for(mov(0,i);i<10;inc(i))do
stdout.put(„i,j=”,i,” „,j,nl);dec(j);endfor;
end masodik;
begin demo;
elso(); masodik();
stdout.put(„i=”,i,”   j=”,j,nl);
end demo;
```

Paraméter átadás

- VAL: Procedure demo_val(N:uns32);
  - demo_val(10);//konstans
  - demo_val(eax);//32 bites regiszter
  - demo_val(uns32_valtozo);// nem fog
  - demo_val(dword_valtozo);// megváltozni
- VAR: Procedure demo_var(VAR N:uns32);
  - demo_var(uns32_valtozo);// “meg fog”
  - demo_var(dword_valtozo);// változni
  - közvetlenül a címét kapjuk meg
- RESULT: Procedure demo_result(RESULT N:uns32);
  - demo_var(uns32_valtozo);// “meg fog”
  - demo_var(dword_valtozo);// változni
  - lokális másolatot kapunk, kezdőérték nélkül
- VALRES: Procedure demo_valres(VALRES N:uns32);
  - demo_var(uns32_valtozo);// “meg fog”
  - demo_var(dword_valtozo);// változni
  - lokális másolatot kapunk, kezdőértékkel

Függvények, visszatérési érték

- FÜGGVÉNYEK: Olyan eljárások, melyek „fő feladata”, hogy egy konkrét „függvényértéket” meghatározzanak -kiszámoljanak-, és azt „visszadják” a hívónak.
- Kompozit utasítások
- A „@returns” opció
- procedure betu_e(c:char);@returns(„eax”); ....
  - mov(betu_e(al),ebx);
  - if(betu_e(chr)) then ... endif;
    - betu_e(chr); if(eax) then ... endif;

Eljárások opciói

- @forward; @noframe; @nodisplay; @noalignstack; @external; @use reg32; @cdecl; @stdcall; @pascal
- Paraméterátadás helye:
  - regiszter
  - verem
  - kód
- Aktivációs Rekord
- CALL és RET utasítások
- @external;#include();// „nagy” programok,
- @external
  - csak a globális szinten
  - csak eljárás, static, readonly, storage
  - static c:char;@external(„var_c”);
- Lokális változók „igazítása”
  - var [4:1]//
  - var {[max{:min}]}{:=start}

```
procedure AddandZero( var p1_ref: uns32; var p2_ref:uns32 )
   ;@nodisplay;@noframe;
var p1: uns32; p2: uns32;
begin AddandZero;
push( ebp ); sub( _vars_, esp ); // Note: _vars_ is "8" in this example.
push( eax );
mov( p1_ref, eax ); mov( [eax], eax ); mov( eax, p1 );
mov( p2_ref, eax ); mov( [eax], eax ); mov( eax, p2 ); pop( eax );
// Actual procedure body begins here:
mov( p2, eax );
add( eax, p1 );
mov( 0, p2 );
// Clean up code associated with the procedure?s return:
push( eax ); push( ebx );
mov( p1_ref, ebx ); mov( p1, eax ); mov( eax, [ebx] );
mov( p2_ref, ebx ); mov( p2, eax ); mov( eax, [ebx] );
pop( ebx ); pop( eax ); ret( 8 );
end AddandZero;//-----------------------------------------
```

```
procedure uhoh( valres i:int32; valres j:int32 ); @nodisplay;
begin uhoh;
mov( 4, i ); mov( i, eax ); add( j, eax );
stdout.put( "i+j=", (type int32 eax), nl );
end uhoh;
...
var k: int32;
...
mov( 5, k ); uhoh( k, k );
...
procedure DisplayAndClear( val i:int32 ); @nodisplay; @noframe;
begin DisplayAndClear;
push( ebp ); // NOFRAME, so we have to do this manually.
mov( esp, ebp );
stdout.put( "I = ", i, nl );
mov( 0, i );
pop( ebp );
ret(); // Note that we don?t clean up the parameters.
end DisplayAndClear;
...
push( m );
call DisplayAndClear;
pop( m );
stdout.put( "m = ", m, nl );
```

Display:

```
procedure dummy(a:int64;b:int64);
var aa:int16;//[ebp-10];
     procedure dm(a:int8;b:int16);
     var x:int8;//[ebp-13]
         y:int16;//[ebp-15]
     begin dm;//push(ebp);push([ebp-4]);push([ebp-8]);
              //lea([esp+8],ebp);push(ebp);sub(4,esp);
              //and($FFFF_FFFC,esp);
     mov(y,ax);
     add(x,al);
     end dm; //mov(ebp,esp);pop(ebp);ret(8);
var bb:int8;//[epb-11]
begin dummy;//push(ebp);push([ebp-4]);lea([esp+4],ebp);
            //push(ebp);sub(4,esp);and($FFFF_FFFC,esp);
mov(aa,ax);
add(bb,al);
end dummy;//mov(ebp,esp);pop(ebp);ret(16);
```

Sztringkezelő utasítások

Adatmozgató utasítások

- LODSB; LODSW; LODSD; // ESI // direction flag
- STOSB; STOSW; STOSD; // EDI
- MOVSB; MOVSW; MOVSD; // ESI, EDI

hasonlító utasítások

- SCASB; SCASW; SCASD; // AL-[EDI]
- CMPSB; CMPSW; SMPSD; // [ESI]-EDI]

Ismétlő prefixek // ECX

- REP.MOZGATO
- REPE.HASONLITO
- REPNE.HASONLITO

HLA sztringjei

- [max-hossz; hossz; maga asztring; lezáró 0; üres]
- str.strRec.length, str.strRec.MaxStrLen
- stralloc; mov(eax,str_var); .... ; strfree(str_var);
- stdin.gets(str_var); std_in.a_gets(); mov(eax,str_var2);