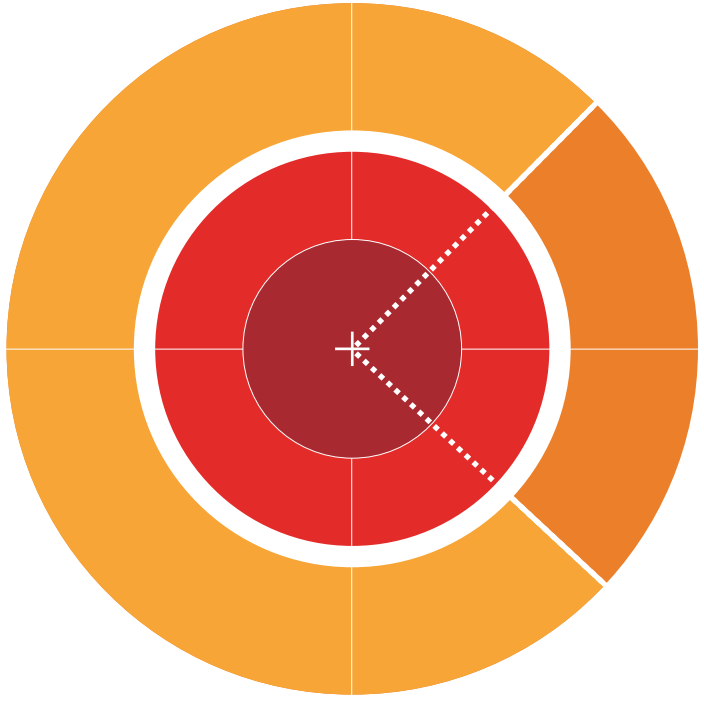
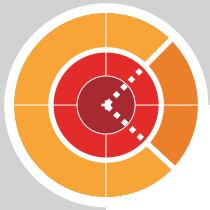


Připojování externích modulů

- často je vhodné využít moduly naprogramované a přeložené v překladačích jiných jazyků, v případě programování v C **typicky v assembleru** (z důvodů rychlosti, nízkoúrovňové obsluhy HW zařízení, apod.)
- obvykle nutně vede k omezení (či úplnému znemožnění) přenositelnosti mezi platformami
- přesný postup se liší podle použitého překladače C a také **linkeru** (v dokumentaci ke každému překladači C by měl být přesný postup popsán)
- odlišnosti jsou zejména v modifikátorech volání funkcí (`near`, `far`, `cdecl`, `pascal`, apod.), v tzv. paměťovém modelu (flat), práci se zásobníkem (kolik parametrů a jakých datových typů se předává subrutinám), atd.
- na dalších slidech bude problematika demonstrovaná na Open Watcom C/C++ 1.5, tj. může se lišit od jiných nástrojů



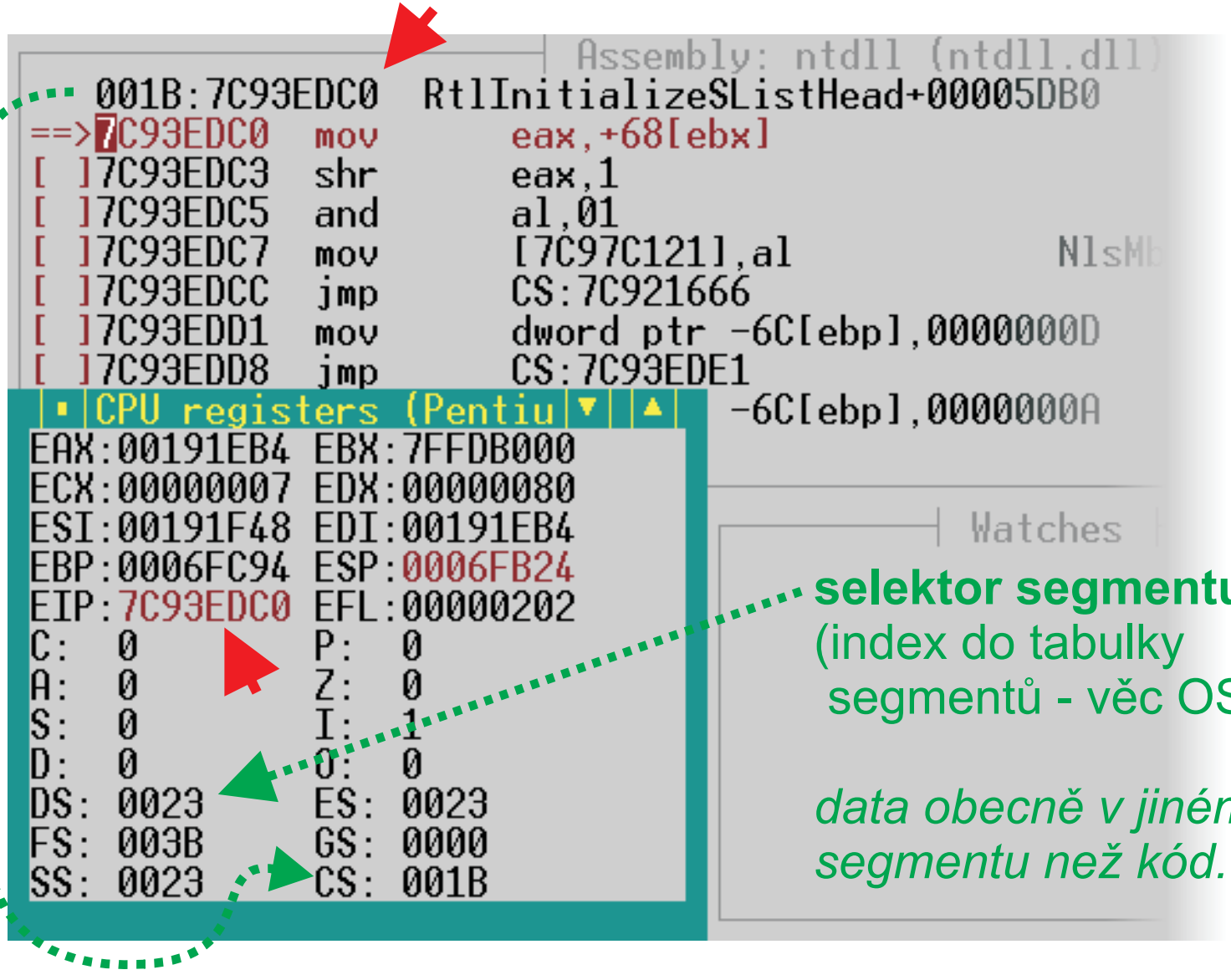
Příprava externího modulu v assembleru

- v Open Watcom assembler **`wasm`**
- důležitá je shoda paměťového (memory) modelu hlavního programu v C a modulu - model určuje, jak se adresuje paměť (jaká je velikost pointeru) a jak je paměť rozdělená na **segmenty**

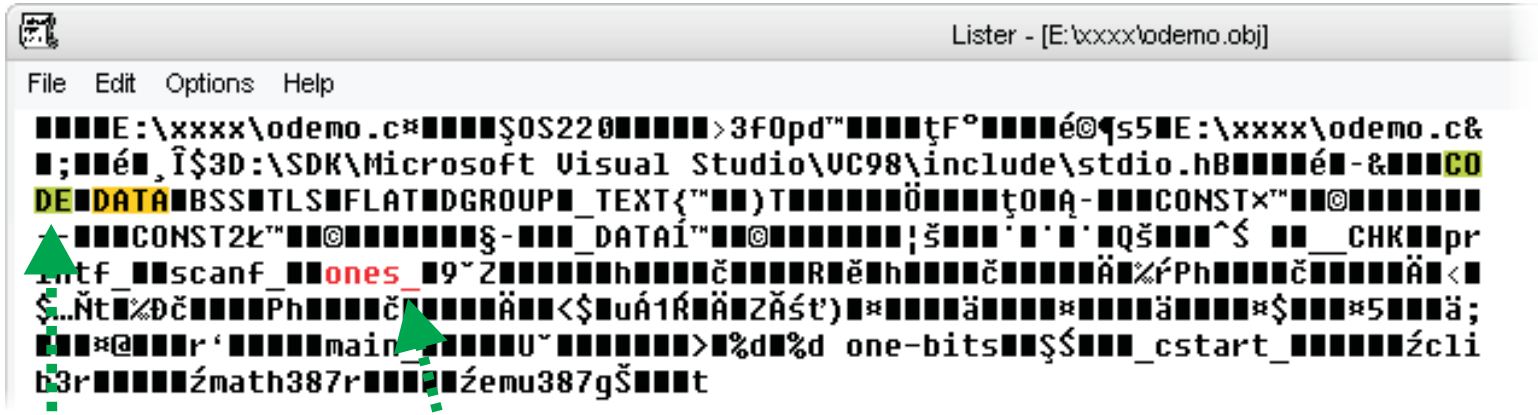
Seg. registr	Obvyklé označení segmentu
CS	CODE
DS	DATA
ES	?
FS	?

- registr CS obsahuje adresu (selektor) segmentu, kde je kód programu; DS totéž pro data (v reálu složitější)
- linker musí vědět, do kterého segmentu připojit kterou část externího .OBJ modulu

Segmentové registry



Pojmenování segmentů, segment v assembleru




```
.386
_TEXT     SEGMENT BYTE PUBLIC 'CODE'
          ASSUME CS:_TEXT
```

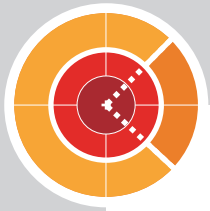
Zdrojový kód v assembleru (styl Intel/MASM)

```
.386
_TEXT       SEGMENT BYTE PUBLIC 'CODE'
            ASSUME CS:_TEXT
            public ones_

ones_       proc near
            mov  cx, 32
            xor  bx, bx
  L1:       shr  eax, 1
            jnc  L2
            inc  bx
  L2:       loop L1
            mov  ax, bx
            ret
ones_       endp

_TEXT       ends
END
```

takto funkci vidí **linker**

používá se pouze 32-bit offset v rámci jednoho segmentu (bez selektoru)

použití všeobecných registrů je bezpečné, překladač je před voláním funkce uloží

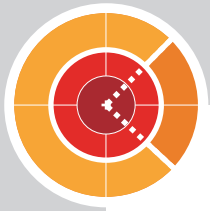


Překlad externího modulu a spojení s programem

```
#include <stdio.h>

extern int ones(int val);

int main() {
  int x;

  do {
    printf(">");
    scanf("%d", &x);
    if (x)
      printf("%d one-bits\n", ones(x));
  } while (x);
  return 0;
}
```

prototyp říká překladači, jak naložit s obsahem registrů a zásobníku (interpretace parametrů)

```
>wasm extfunc.asm
>wcl386 program.c extfunc.obj
```



Předávání parametrů podprogramům (snazší případ)

```
Assembly: stacktest2
   001B:00401057  main+00000018
            func(1, 2, 3);
[ ]00401057  mov       ebx,00000003
[ ]0040105C  mov       edx,00000002
[ ]00401061  mov       eax,00000001
[ ]00401066  call      func
            return 0;
[ ]0040106B  mov       dword ptr -4[ebp],00000000
   }
```

předávání konstant (hodnotou) je snadné... **hodnoty se umístí přímo do** obecných **registrů**

- pořadí obsazování obecných střadačů:
  (i) **EAX**, (ii) **EDX**, (iii) **EBX**, (iv) **ECX**

- pokud se předávají proměnné, jejich hodnota se taky umístí do registrů, ale zároveň se musí v zásobníku vytvořit jejich lokální kopie, aby s nimi mohla funkce pracovat...

Předávání parametrů podprogramům (složitější případ)

```
Assembly: stacktest
   001B:0040104B  main
            int a0 = 0, a1 = 1, a2 = 2, a3 = 3, a4 = 4;
[ ]00401063  mov       dword ptr -14[ebp],00000000
[ ]0040106A  mov       dword ptr -10[ebp],00000001
[ ]00401071  mov       dword ptr -0C[ebp],00000002
[ ]00401078  mov       dword ptr -8[ebp],00000003
[ ]0040107F  mov       dword ptr -4[ebp],00000004
            func(a0, a1, a2, a3, a4);
[ ]00401086  mov       eax,-4[ebp]
[ ]00401089  push      eax
[ ]0040108A  mov       ecx,-8[ebp]
[ ]0040108D  mov       ebx,-0C[ebp]
[ ]00401090  mov       edx,-10[ebp]
[ ]00401093  mov       eax,-14[ebp]
[ ]00401096  call      func
            return 0;
[ ]0040109B  mov       dword ptr -18[ebp],00000000
```

v zásobníku vznikají **lokální** kopie předaných proměnných

když dojdou obecné střadače, umisťují se parametry do zásobníku "odspodu", tj. 1. parametr nejhlouběji...

Předávání reálných parametrů (nejsložitější případ)

Assembly: stacktest3

```
   001B:00401041  main
            double x = 2.5, y = 3.5;
[ ]00401059  mov       dword ptr -14[ebp],00000000
[ ]00401060  mov       dword ptr -10[ebp],40040000
[ ]00401067  mov       dword ptr -0C[ebp],00000000
[ ]0040106E  mov       dword ptr -8[ebp],400C0000
            func(x, y);
[ ]00401075  fld       qword ptr -0C[ebp]          0023
[ ]00401078  sub       esp,00000008
             fstp      qword ptr [esp]             0023
             fld       qword ptr -14[ebp]          0023
             sub       esp,00000008
             fstp      qword ptr [esp]             0023
[ ]00401087  call      func
[ ]0040108C  fstp      st(0)
            return 0;
[ ]0040108E  mov       dword ptr -4[ebp],00000000
   }
```

instrukce koprocesoru 80x87

registr koprocesoru

Předávání reálných parametrů (nejsložitější případ)

```
   001B:00401
            do
[ ]00401059
[ ]00401060
[ ]00401067
[ ]0040106E
            func(x, y);
[ ]00401075   fld       qword ptr -0C[ebp]
[ ]00401078   sub       esp,00000008
              fstp      qword ptr [esp]
              fld       qword ptr -14[ebp]
              sub       esp,00000008
[ ]00401087
[ ]0040108C
            ret
[ ]0040108E
   }
```

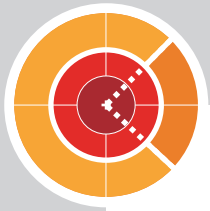
instrukce koprocesoru 80x87

FLD—Load Real intel

Opcode	Instruction	Description
D9 /0	FLD *m32real*	Push *m32real* onto the FPU register stack.
DD /0	FLD *m64real*	Push *m64real* onto the FPU register stack.
DB /5	FLD *m80real*	Push *m80real* onto the FPU register stack.
D9 C0+i	FLD ST(i)	Push ST(i) onto the FPU register stack.

FST/FSTP—Store Real intel

Opcode	Instruction	Description
D9 /2	FST *m32real*	Copy ST(0) to *m32real*
DD /2	FST *m64real*	Copy ST(0) to *m64real*
DD D0+i	FST ST(i)	Copy ST(0) to ST(i)
D9 /3	FSTP *m32real*	Copy ST(0) to *m32real* and pop register stack
DD /3	FSTP *m64real*	Copy ST(0) to *m64real* and pop register stack
DB /7	FSTP *m80real*	Copy ST(0) to *m80real* and pop register stack
DD D8+i	FSTP ST(i)	Copy ST(0) to ST(i) and pop register stack



Zpracování parametrů uvnitř podprogramu (assembler)

```
Assembly: stackte
   001B:00401015  func+00000005
   long int func(long int a, long int b) {
[ ]00401010  push      00000024
==>00401015  call      __CHK
[ ]0040101A  push      ebx
[ ]0040101B  push      ecx
[ ]0040101C  push      esi
[ ]0040101D  push      edi
[ ]0040101E  push      ebp
[ ]0040101F  mov       ebp,esp
[ ]00401021  sub       esp,0000000C
[ ]00401027  mov       -0C[ebp],eax
[ ]0040102A  mov       -8[ebp],edx
            return a + b;
[ ]0040102D  mov       eax,-0C[ebp]
[ ]00401030  add       eax,-8[ebp]
[ ]00401033  mov       -4[ebp],eax
   }
[ ]00401036  mov       eax,-4[ebp]
[ ]00401039  mov       esp,ebp
[ ]0040103B  pop       ebp
```

optimalizace!

kontrolní mechanismus překladače

překladač ihned po vstupu do podprogramu uschová hodnoty nevyužívaných registrů

(to my při programování ASM rutin nemusíme - víme, co si přepíšeme a co ne...)

vyzvednutí parametrů ze zásobníku, provedení výpočtu, uložení výsledku do zásobníku...

návratová hodnota je v EAX (pokud se tam vejde)