UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Sisteme de Operare 2

9. Asamblare și Linking

Sintaxa gas (AT&T)

- Comentarii
  - –Pe mai multe linii: /* */
  - –Pana la sfarsitul linei (dependente de platforma): # ! ; |
- Operanzi:
  - Imediati: precedati de $
    - –Intel: push 4
    - –AT&T: push $4
  - Registri: precedati de %
    - –Intel: push eax
    - –AT&T: push %eax

Sintaxa gas (AT&T) (2)

- Operanzi (2)
  - Sursa si destinatia sunt inversate:
    - –Intel: add eax, 4
    - –AT&T: add $4, %eax
  - Dimensiunea operandului este specificata prin postfixarea b, w, l, q la mnemonica instructiunii:
    - –Intel: mov al, byte ptr FOO
    - –AT&T: movb FOO, %al

Sintaxa gas (AT&T) (3)

- Adresare indirectă
  - Intel: [base + index*scale + displ]
  - AT&T displ(base, index, scale)
- Exemple
  - –Intel: mov eax, [100]
  - –AT&T: mov 100(,1), %eax
  - –Intel: mov eax, [ebx-100]
  - –AT&T: mov -100(%ebx), %eax
  - –Intel: mov eax, [100+ebx*4]
  - –AT&T: mov 100(,%ebx,4), %eax

Frame stack

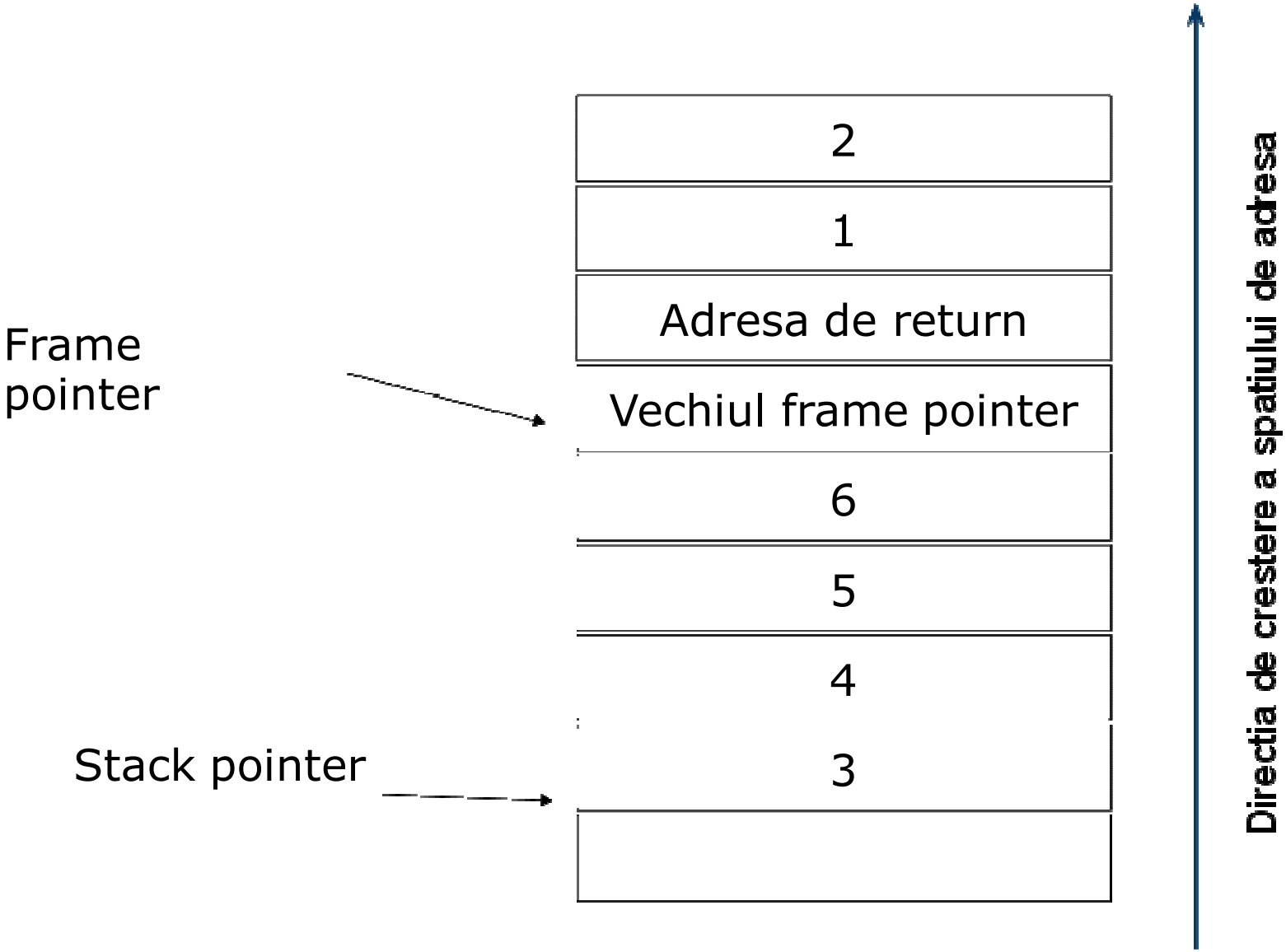
- Modul în care sunt aranjate pe stivă informațiile pentru funcția curentă
- Stiva programului este o stivă de frame-uri
- Frame stack-ul conține:
  - Parametrii funcției
  - Adresa de return
  - Opțional: adresa vechiului frame pointer
  - Variabilele locale

Exemplu

```
int a(int b, int c)
{
     int d=3, e=4, f=5, g=6;
     return b+c+d+e+f+g;
}

int main()
{
    return a(1,2);
}
```

Stack frame-ul pentru funcția a



Stack frame-ul pentru funcția a (2)

```
(gdb) break 4
Breakpoint 1 at 0x8048366: file f.c, line 4.
(gdb) run
Starting program: /home/tavi/a.out
Breakpoint 1, a (b=1, c=2) at f.c:4
4 return b+c+d+e+f+g;
(gdb) x/8 $esp
0xbfecae94: 0x00000003 0x00000004 0x00000005 0x00000006
0xbfecaea4: 0xbfecaeb8 0x0804839f 0x00000001 0x00000002
```

Invocarea unei funcții

```
$objdump -dr a.o
....
44: sub $0x8,%esp
47: movl $0x2,0x4(%esp)
4f: movl $0x1,(%esp)
56: call 57 <main+0x21>
57: R_386_PC32 a
...
```

Accesul la parametri şi variabile locale

```
00000000 <a>:
0: push %ebp
1: mov %esp,%ebp
3: sub $0x10,%esp
6: movl $0x3,-0x10(%ebp)
d: movl $0x4,-0xc(%ebp)
14: movl $0x5,-0x8(%ebp)
1b: movl $0x6,-0x4(%ebp)
22: mov 0xc(%ebp),%eax
25: add 0x8(%ebp),%eax
28: add -0x10(%ebp),%eax
2b: add -0xc(%ebp),%eax
2e: add -0x8(%ebp),%eax
31: add -0x4(%ebp),%eax
34: leave
35: ret
```

Exemplu de invocare

```
000c7590 <__dup2>:
c7590: mov %ebx,%edx
c7592: mov 0x8(%esp),%ecx
c7596: mov 0x4(%esp),%ebx
c759a: mov $0x3f,%eax
c759f: int $0x80
c75a1: mov %edx,%ebx
```

GCC extented asm

- Folosirea de cod ASM împreună cu cod C
- Se pastrează claritatea codului
- Se “optimizează de mâna”
- Se accesează resurse altfel neaccesibile din compilator (regiştri speciali, instrucțiuni speciale)

asm / __asm__

```
asm(intruction_template
    : constraints_1 output_operand_1,
     constraints_2 output_operand_2,
     ...,
     constraints_n output_operand_n
    : constraints_n+1 input_operand_n+1,
     constraints_n+2 input_operand_n+2,
     ...,
     constraints_n+m input_operand_n+m,
    : clobbered_regmem_1, clobbered_regmem_2,
     ...,
     clobbered_regmem_n);
```

Instruction template

- instruction_mnemonic [operand_asm | operand_no], ...
  - operand_asm = „%%registru”
  - operand_no = „%x” unde x este numărul operandului

```
unsigned long address;
asm(“mov %%cr2, %0”: (address));
printk(“Adress: 0x%x”, address);
```

Constrângeri

- “m” = orice fel de operator memorie
- “r” = orice fel de operator registru
- “i” = un intreg imediat (intreg cu valoarea cunoscuta la momentul asamblarii)
- “0”, “1”, …, “9” = impune sa se foloseasca acelasi operand ca cel indicat
- “=” = operandul este write-only; se foloseste pentru operanzii de output

Exemplu

```
void printk_cr2()
{
    unsigned long address;
    asm(“mov %%cr2, %0”: "=m" (address));
    printk(“0x%x”, address);
}
```

Exemplu (2)

```
00000000 <printk_cr2>:
0: push %ebp
1: mov %esp,%ebp
3: sub $0x18,%esp
6: mov %cr2,%eax
9: mov %eax,-0x4(%ebp)
c: mov -0x4(%ebp),%eax
f: mov %eax,0x4(%esp)
13: movl $0x0,(%esp)
16: R_386_32 .rodata
1a: call 1b <x+0x1b>
1b: R_386_PC32 printk
1f: leave
20: ret
```

Efecte laterale

- Unele instrucțiuni pot modifica şi alți regiştri / zone de memorie decât operanzii specificați
- Operanzii clobbered_regmem sunt folosiți pentru a indica compilatorului care regiştri / zone de memorie sunt afectate de blocul asm
- (blocul asm şi codul generat de compilator pot folosi acelaşi registru zonă de memorie)