Aufgabenblatt 12

Ausgabe 18/01/2010, Abgabe bis 25/01/2010 12:00

Name(n):
Matrikelnummer(n):
Übungsgruppe:

Aufgabe 12.1 Bonus (15 Punkte)

Bonus wegen verspäteter Bereitstellung des Aufgabenblattes :-)

Aufgabe 12.2 lea-Befehl (15 Punkte)

Der Befehl `lea expr, dest` („load effective address"') ist eine Besonderheit der x86-Architektur. Der Befehl berechnet den arithmetischen Ausdruck `expr` gemäß $\mathrm{imm} + x + k \cdot y$ mit einem (optionalen) Immediatewert imm, zwei Operanden x und y und einem (optionalen) Skalierungsfaktor $k = 1, 2, 4, 8$ und speichert das Ergebnis im angegeben Ziel `dest`. Der Befehl dient eigentlich der Berechnung von Speicheradressen (und verwendet dasselbe Rechenwerk wie die Speicheradressierung), wird aber von Compilern gerne auch zur Berechnung von arithmetischen Ausdrücken verwendet.

Die folgende Tabelle enthält verschiedene leal-Befehle, die das Resultat jeweils im Register `%edx%` ablegen. Geben Sie die Formeln an, welche arithmetischen Ausdrücke den folgenden lea-Befehlen entsprechen:

lea-Befehl	Wert entspricht
`leal (%eax,%ecx,8), %edx`	$x + 8y$
`leal 7(%eax), %edx`	
`leal (%ebx,%ecx), %edx`	
`leal 5(%eax,%eax,4), %edx`	
`leal $0x1A(,%eax,2), %edx`	
`leal 9(%eax,%ecx,2), %edx`	

Aufgabe 12.3 PC-relative Addressierung (10 + 10 + 10 Punkte)

Die x86-Architektur erlaubt bei Sprungbefehlen (`call`, `jmp`, `je` und Varianten) sowohl die Angabe absoluter Zieladressen, als auch die Berechnung relativ zum aktuellen Wert des Program-Counters `eip`. Diese verschiedenen Möglichkeiten werden als separate Befehle mit unterschiedlichen Opcodes codiert.

Bei PC-relativen Sprüngen wird der Offset vorzeichenbehaftet mit 1, 2, oder 4 Bytes kodiert, und wird relativ zur Startadresse des nachfolgenden (!) Befehls angegeben. (Dieses Verhalten ist darauf zurückzuführen, dass ältere x86-Prozessoren den Wert des Registers `eip` als ersten Schritt der Befehlsausführung inkrementierten.)

Überlegen Sie sich in den folgenden Beispielen die relevanten Adressen und ersetzen Sie die Platzhalter `xxxxxxx` jeweils durch die passenden Werte:

a) Was ist die Zieladresse des Befehls `jbe` (jump if below or equal) im folgenden Beispiel (Opcode 0x76, Offset 0xda im Zweierkomplement):

```
804001c:   76 da            jbe  xxxxxxx
804001e:   eb 24            jmp  8040044
```

b) Ergänzen Sie die Adressen:

```
xxxxxxx:   eb 54            jmp  8050d42
xxxxxxx:   c7 45 f8 10 00   mov  $0x10,0xfffffff8(%ebp)
```

c) Ergänzen Sie die Sprungadresse (4-Byte Offset, Byte-Order beachten):

```
8040000:   e9 cb 00 00 00   jmp  xxxxxxx
8040005:   90               nop
```

Aufgabe 12.4 Rekursion (30 + 10 Punkte)

Analysieren Sie das folgende in C-Syntax notierte Programm, und geben Sie einen mathematischen Ausdruck für das Ergebnis der Funktion an:

```
int wsdw(int a, int b)
{
    if (b == 0) {
                return 1;
    }
    else {
                return wsdw(a, b-1) * a;
    }
}
```

Für die Variablen a, b gilt die Beschränkung: a,b ≥ 0.

a) Schreiben Sie ein rekursives x86-Assemblerprogramm, das die Funktion des obigen C-Programms erfüllt. Die Parameter a und b werden durch die rufende Prozedur/Funktion auf dem Stack hinterlegt. Der Resultatwert soll gemäß Konvention wieder in dem Register `%eax` hinterlegt werden.

b) Fertigen Sie eine Skizze des Stacks bei maximaler Verschachtelungstiefe an. Verwenden sie dabei für den ersten Aufruf von `wsdw` die Werte $a = 5$ und $b = 3$ für die Parameter.