64-041 Übung Rechnerstrukturen

Aufgabenblatt 12 Ausgabe: 18.01., Abgabe: 25.01. 12:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 12.1 (Punkte 5+5+5+5+5+5)

x86-Assembler: Angenommen, die folgenden Werte sind in den angegebenen Registern bzw. Speicheradressen gespeichert:

Register	Wert
%eax	0x00000100
%ecx	0x0000000C
%edx	0x00000004

Adresse	Wert
0x100	0x0000CAFE
0x104	0x000000CB
0x108	0x00012300
0x10C	0x00000042

Überlegen Sie sich, welche Speicheradressen bzw. Register als Ziel der folgenden Befehle ausgewählt werden und welche Resultatwerte sich aus den Befehlen ergeben:

(a) `addl %edx, (%eax)`

(b) `subl %ecx, 4(%eax)`

(c) `imull $32, (%eax,%edx,2)`

(d) `incl 12(%eax)`

(e) `decl %ecx`

(f) `subl %edx, %eax`

Zur Erinnerung: für den gnu-Assembler gilt

- der Zieloperand steht rechts
- Registerzugriffe werden direkt ausgedrückt
- eine runde Klammer um ein Register bedeutet einen Speicherzugriff, ggf. mit Immediate-Offset und Index: $\langle imm \rangle$`(`$\langle Rb \rangle$`,`$\langle Ri \rangle$`,`$\langle s \rangle$`)` $\rightarrow$ `MEM[`$\langle Rb \rangle$`+`$\langle s \rangle$`*`$\langle Ri \rangle$`+`$\langle imm \rangle$`]`

$\Rightarrow$ zum Beispiel bewirkt der Befehl: `addl %edx, 8(%eax)`
die Operation: `MEM[0x00000108] = 0x00012304`

Sie können die Befehle natürlich gerne auch im Assembler und Debugger direkt ausprobieren. Mit einigen Befehlen lassen sich die oben angegebenen Werte in den Speicher schreiben, und die Resultate lassen sich dann direkt ablesen. Geben Sie in diesem Fall Ihr Assemblerprogramm bitte mit ab.

Aufgabe 12.2 (Punkte 5)

Register auf Null setzen: Wie kann man den Inhalt eines Registers auf Null setzen, wenn dafür kein separater Befehl zur Verfügung steht? Geben Sie x86-Beispielcode an, der *ohne Immediate-Operand* auskommt.

Aufgabe 12.3 (Punkte 10)

Programmzähler auslesen: Es gibt keinen x86-Assemblerbefehl, der es erlaubt, den Programmzähler `%eip` direkt auszulesen. Schreiben Sie ein kurzes Assemblerprogramm, das den Programmzähler in das Register `%eax` kopiert. Hinweis: Sie dürfen den Stack zur Zwischenspeicherung verwenden.

Aufgabe 12.4 (Punkte 20)

Arithmetische Operationen: Eine klassische Aufgabe zur Demonstration einfacher numerischer Operationen ist die Umrechnung zwischen Grad Fahrenheit F und Grad Celsius C nach der Formel $C = (F - 32) * 5/9$.

Da im bisher eingeführten x86-Befehlssatz noch kein Befehl für die Division enthalten ist, nähern wie den Umrechnungsfaktor $5/9$ durch den Wert $5/9 \approx 142/256$ an, der sich zum Beispiel mit Multiplikation (`imull` ⟨*src*⟩`,`⟨*dest*⟩) und Rechtsschieben (`sarl` bzw. `shrl` für arithmetisches und logisches Schieben) effizient umsetzen lässt.

Schreiben Sie x86-Assemblercode für eine Funktion `int f2c (int f)`, die ihr Argument (Grad Fahrenheit), wie in der Vorlesung erläutert, auf dem Stack übergeben bekommt und ihren Rückgabewert entsprechend der Konvention im Register `%eax` hinterlässt.

Nach Ausführung der Funktion sollen die relevanten Datenregister wieder ihren vorherigen Wert enthalten. Bedenken Sie dabei, dass laut Konvention die Register `%eax`, `%edx` und `%ecx` als „Caller-Save“ klassifiziert sind. Daraus ergibt sich, dass Inhalte der für die Berechnung benötigten Register von der Funktion teilweise ebenfalls auf den Stack gerettet und am Ende wiederhergestellt werden müssen.

Aufgabe 12.5 (Punkte 10+5+15+5)

x86-Assembler entschlüsseln: Wir betrachten die folgende Funktion `int myst (int a, int b)`, die zwei 32-bit Integer-Parameter entgegennimmt und einen 32-bit Integerwert im Register `eax` zurückliefert:

```
myst:
        pushl   %ebx
        subl    $24, %esp
        movl    32(%esp), %ebx
        movl    36(%esp), %edx
        movl    $1, %eax
        testl   %edx, %edx
        je      .L2
        subl    $1, %edx
        movl    %edx, 4(%esp)
        movl    %ebx, (%esp)
        call    myst
        imull   %ebx, %eax
.L2:
        addl    $24, %esp
        popl    %ebx
        ret
```

(a) Kommentieren Sie die einzelnen Assemblerbefehle. Zur Erinnerung: C legt die Parameter beim Aufruf *von rechts nach links* auf den Stack.

(b) Was berechnet das Unterprogramm?

(c) Wie sieht der Stack bei maximaler Verschachtelungstiefe für den Aufruf von `myst (5, 2)` aus? Ergänzen Sie die Skizze.

	...	alter Stack von `main()`
	b=2	Parameter für Aufruf von `myst(5, 2)`
	a=5	–"–
`%esp` →	`%eip main()`	`call`-Befehl

(d) Was würde bei einem Aufruf von `myst (3, -3)` geschehen?