Aufgabenblatt 11

Ausgabe 17/01/2011, Abgabe bis 24/01/2011 12:00

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

Aufgabe 11.1 x86-Assembler (5+5+5+5+5+5 Punkte)

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

Register	Wert
`%eax`	`0x00000100`
`%ecx`	`0x00000001`
`%edx`	`0x0000000C`

Adresse	Wert
`0x100`	`0x0000BEEF`
`0x104`	`0x000000AC`
`0x108`	`0x00000013`
`0x10c`	`0x00078900`

Überlegen Sie sich, welche Adressen bzw. Register als Ziel der folgenden Befehle ausgewählt werden und welche Resultatwerte sich ergeben:

`Befehl`	Ziel (Adresse/Register)	Wert
`addl %ecx, (%eax)`		
`subl %edx,4(%eax)`		
`imull $16,(%eax,%edx)`		
`incl 8(%eax)`		
`decl %ecx`		
`subl %edx,%eax`		

Hinweis:
Beim gnu-Assembler steht der Zieloperand rechts, und eine runde Klammer um ein Register bedeutet einen Speicherzugriff auf die entsprechende Adresse, ggf. mit dem vor der Klammer notierten Byte-Offset. Zum Beispiel bewirkt
der Befehl: `addl %ecx,12(%eax)`
die Operation: `MEM[0x0000010c] = 0x00078901`

(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 11.2 x86-Assembler: Register auf Null setzen (5+15 Punkte)

a) 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.

b) Wie kann man die Inhalte von zwei Registern vertauschen, ohne ein zusätzliches Register oder eine zusätzliche Speicherstelle zu verwenden? Geben Sie als Beispiel den x86 Assemblercode an, um die Werte in den Registern %eax und %edx zu vertauschen.

Hinweis:
Denken Sie auch über die XOR-Operation nach. Der x86 Befehl dafür lautet `xorl src, dest`.

Aufgabe 11.3 x86-Assembler: 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 11.4 x86-Assembler: Programmzähler auslesen (15 Punkte)

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 11.5 x86-Assembler: Umrechnung Grad Fahrenheit nach Celsius (20 Punkte)

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

Da im bisher eingeführten Befehlssatz für unseren x86-Prozessor 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 einer Multiplikation (`imull src dest`) und einem Rechts-Shift (`sarl` bzw. `shrl` für arithmetisches und logisches shift-right) effizient umsetzen lässt.

Schreiben Sie x86-Assemblercode für eine Funktion `int celsius( int fahrenheit )`, die ihr Argument (in 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. Die Inhalte der für die Berechnung benötigten Register müssen also von der Funktion teilweise ebenfalls auf den Stack gerettet und am Ende wiederhergestellt werden.