64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 12 Ausgabe: 17.01., Abgabe: 24.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 12.1 (Punkte 10)

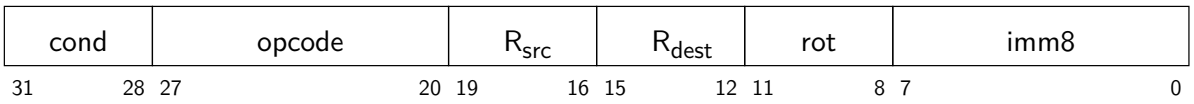
CISC vs. RISC: erläutern Sie beide Begriffe. Was sind typische Eigenschaften der jeweiligen Befehlssätze und Architekturen?

Aufgabe 12.2 (Punkte 4+4+4+4+4)

Darstellung von Immediate-Operanden: Um trotz eingeschränkter Wortlängen bei RISC-Befehlssätzen möglichst viele, häufig benötigte Werte als *Immediate* darzustellen, benutzen die Befehlssätze aktueller Prozessoren einige Tricks. Ein gutes Beispiel zeigt die für eingebettete Systeme und Mobilgeräte sehr beliebte 32-bit ARM-Architektur. Dort ist unter anderem für arithmetische Befehle ein Format mit Immediate-Operanden definiert, bei dem der *Barrel-Shifter* benutzt wird, um 32-bit Immediate-Werte zu erzeugen:

$\langle imm32 \rangle$ = $\langle imm8 \rangle$ *rotate-right* ($\langle rot \rangle \cdot 2$)

$\langle imm8 \rangle$	8-bit	0…255	beliebiger Ausgangswert
$\langle rot \rangle$	4-bit	0…15	Distanz der *rotate-right* Operation: $\langle rot \rangle \cdot 2$ (Schrittweite 2)



Überlegen Sie sich die jeweilige 12-bit Codierung ($\langle rot \rangle$ $\langle imm8 \rangle$) der folgenden Immediate-Werte oder begründen Sie, warum ein Wert nicht dargestellt werden kann.

(a) 207

(b) 398

(c) 351

(d) 1 212 416

(e) 2 415 919 104

Aufgabe 12.3 (Punkte 6·5)

x86-Adressierung: Angenommen, die folgenden Werte sind in den angegebenen Registern bzw. Speicheradressen gespeichert. Adressen sind verkürzt dargestellt und von den Werten sind jeweils auch nur die unteren 32-bit angegeben:

Register	Wert
`%rax`	`0x00000100`
`%rcx`	`0x00000020`
`%rdx`	`0x00000018`

Adresse	Wert
`0x100`	`0x0000abcd`
`0x108`	`0x000000ef`
`0x110`	`0x0000beef`
`0x118`	`0x00054321`

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

Befehl	Ziel (Adresse/Register)	Wert
`addq %rcx, (%rax)`		
`subq %rdx, 8(%rax)`		
`imulq $16, %rdx`		
`incq 16(%rax)`		
`decq %rcx`		
`subq %rdx, %rax`		

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[`$\langle Rb \rangle$`+`$\langle s \rangle$`*`$\langle Ri \rangle$`+`$\langle imm \rangle$`]`

Beispiel: Befehl `addq %rcx, 24(%rax)`
Operation `MEM[0x00000118] := MEM[0x00000118]+0x00000020 = 0x00054341`

Aufgabe 12.4 (Punkte 3·5)

Flags: Bei dieser Aufgabe sollen Sie sich überlegen, welche Flags (Carry, bzw. Overflow) durch arithmetische Operationen in der CPU gesetzt werden. Kreuzen Sie die Flags an und begründen Sie Ihre Entscheidung kurz. Dabei sollen auch die Dezimalwerte der Zahlen (32-bit, 2-Komplement) mit angegeben werden.

Operation	Inhalt `%eax`	Inhalt `%ebx`	`CF`	`OF`
`addl %eax, %ebx`	`0x00000002`	`0x7fffffff`		
`addl %eax, %ebx`	`0x80000000`	`0xfffffffe`		
`addl %eax, %ebx`	`0x00000001`	`0xffffffff`		

Anmerkung: um nicht mit 64-bit Zahlen zu rechnen, sind die Register, deren Inhalte und die Operationen jeweils 32-bit breit. Bei der x86-Architektur werden die Register `%e..` statt `%r..` benutzt (s. Folie 870) und die Assemblerbefehle sind `addl` statt `addq` (*long* statt *quad*).

Zur Erinnerung: das Carry-Flag wird bei einer arithmetischen Operation gesetzt, wenn sich in der höchstwertigsten Stelle ein Übertrag ergibt. Das Overflow-Flag wird gesetzt, wenn das Ergebnis der Operation das „falsche“ Vorzeichen hat, beispielsweise wenn die Addition zweier positiver Zahlen ein negatives Ergebnis liefert.

Aufgabe 12.5 (Punkte 10+15)

x86-Prozeduraufrufe und Stack: Der Stack ist für die Ausführung von Funktionen, bzw. Prozeduren von zentraler Bedeutung.

(a) In der Vorlesung wurden Befehle vorgestellt, die den Stackpointer `%rsp` indirekt verändern, d.h. ohne dass `%rsp` als Argument in dem Befehl vorkommt. Welche Assemblerbefehle sind das? Zählen Sie diese Befehle auf, jeweils mit einer kurzen Beschreibung was sie bewirken.

(b) Bei Unterprogrammaufrufen werden auf dem Stack verschiedene Daten gespeichert: sowohl zur Vorbereitung des Aufrufs, als auch während dessen Abarbeitung. Dabei können bis zu 5 „Arten“ von Daten auf dem Stack abgelegt werden. Welche sind das? (Aufzählung mit kurzer Beschreibung).
Tipp: Überlegen Sie sich den Ablauf eines Unterprogrammaufrufs.