# 64-041 Übung Rechnerstrukturen WS 15/16

## Aufgabenblatt 11 Ausgabe: 06.01., Abgabe: 13.01. 24:00

| Gruppe | |
|---|---|
| Name(n) | Matrikelnummer(n) |
| | |

### Aufgabe 11.1 (Punkte 3+3+3+3+3)

*Adressierung*

Welcher Wert steht nach Ausführung der folgenden Befehle im Akkumulator einer 1-Adress-Maschine?

**a)** `LOAD IMMEDIATE 20`
**b)** `LOAD DIRECT 20`
**c)** `LOAD INDIRECT 20`
**d)** `LOAD DIRECT 30`
**e)** `LOAD INDIRECT 30`

Der Hauptspeicher enthält jeweils diese Werte:

| Adresse | Inhalt |
|---|---|
| 20 | 50 |
| 30 | 40 |
| 40 | 60 |
| 50 | 70 |

### Aufgabe 11.2 (Punkte 15)

*Befehlskodierung*

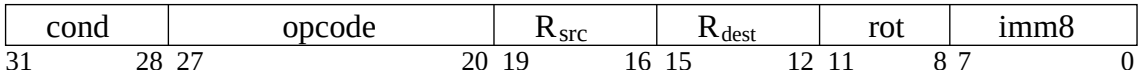
Entwerfen Sie eine Befehlskodierung, um alle der folgenden Befehle in 36-bit Befehlsworten unterzubringen:

| | |
|---|---|
| 7 | Befehle mit zwei 15-bit Adressen und einer 3-bit Registernummer |
| 500 | Befehle mit einer 15-bit Adresse und einer 3-bit Registernummer |
| 50 | Befehle ohne Adressen oder Registerangaben |

## Aufgabe 11.3 (Punkte 4+4+4+4+4)

*Darstellung von Immediate-Operanden*

Die für eingebettete Systeme und Mobilgeräte sehr beliebte 32-bit ARM-Architektur verwendet die folgende Darstellung von Immediate-Operanden für die arithmetischen Befehle:




Der bei diesen Befehlen verwendete Immediate-Wert ergibt sich aus folgendem Ausdruck:

Immediate-Wert $= \text{imm8} \cdot 2^{(2*\text{rot})}$

mit $0 \leq \text{imm8} \leq 255$ und $0 \leq \text{rot} \leq 12$.

Überlegen Sie sich die jeweilige 12-bit Kodierung der folgenden (dezimalen) Zahlenwerte, oder begründen Sie, warum ein Wert nicht dargestellt werden kann:

(a) 251

(b) 259

(c) 1233125376 $(= 2^{23} * 147)$

(d) 2684354560 $(= 2^{31} + 2^{29})$

(e) 573440 $(= 2^{14} * 35)$

## Aufgabe 11.4 (Punkte 5+15)

*x86-Assembler*

(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 vertausche, ohne ein zusätzliches Register oder eine zusätzliche Speicherstelle zu verwenden? Geben Sie als Beispiel den x86 Assemblercode an, um die Werte in den Regstern `%eax` und `%edx` zu vertauschen.

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

## Aufgabe 11.5 (Punkte 5+5+5+5+5)

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

| Register | Wert |
|---|---|
| %eax | 0x00000100 |
| %ecx | 0x00000002 |
| %edx | 0x0000000C |

| Adresse | Wert |
|---|---|
| 0x100 | 0x0000ABBA |
| 0x104 | 0x000000DC |
| 0x108 | 0x000000EF |
| 0x10C | 0x00054321 |

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

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

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

(c) `imull $16, (%eax, %edx)`

(d) `incl 8(%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 auf die entsprechende Adresse, ggf. mit Byte-Offset vor der Klammer

⇒ zum Beispiel bewirkt der Befehl: `addl %ecx, 12(%eax)`
die Operation: `MEM[0x0000010C] = 0x00054323`

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.