64-041 Übung Rechnerstrukturen

Aufgabenblatt 11 Ausgabe: 07.01., Abgabe: 14.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 11.1 (Punkte 5+5+5+5+5)

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

Register	Wert
%eax	0x00000104
%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, 0xfffffffc(%eax)`

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

(d) `incl 4(%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[0x0000010C] = 0x00000046`

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 (Punkte 10)

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 für zwei Möglichkeiten zur Lösung dieses Problems an, der *ohne Immediate-Operand* auskommt.

Aufgabe 11.3 (Punkte 10+10+20+20)

x86-Assembler entschlüsseln:
In manchen Kryptologie-Verfahren ist es notwendig, Ausdrücke der Form

$$a^N \mod m \tag{1}$$

mit natürlichen Zahlen a, N und m möglichst effizient zu berechnen. Dabei können und sollten für eine ausreichend hohe Sicherheit alle diese Zahlen 512 Bit und mehr haben. Wir begnügen uns hier mit viel kleineren Zahlen, die sich noch mit der normalen Rechnerarithmetik bearbeiten lassen. Und zwar wählen wir willkürlich

$$a = 10403 \quad N = 500\,000\,002 \quad m = 11119$$

(a) Betrachten wir zunächst eine völlig unbrauchbare Methode zur Berechnung des obigen Ausdrucks, indem wir erst a^N berechnen und dann das Ergebnis modulo m nehmen. Wieviele Dezimalstellen hätte a^N für die oben angegeben Zahlen? Wieviele Stellen hätte diese Zahl im Dualsystem? Hinweis: Die Logarithmus-Funktion dürfte zur Beantwortung dieser Fragen hilfreich sein.

(b) Die Größe des obigen Zwischenergebnisses überzeugt schnell davon, dass man sich etwas anderes überlegen muss. Der erste Gedanke ist wahrscheinlich, dass man nach jeder Multiplikation modulo m rechnet, was die Zwischenergebnisse klein hält. Man rechnet also

```
x= 1;
for (i=1; i <= N; i++)
    x= (x*a) % m;
```

Dass das zum richtigen Ergebnis führt, sollte aus der Mathematikvorlesung klar sein.

Laden Sie sich die Dateien *poti.c* und *poti2.s* herunter. Übersetzen Sie jetzt das Programm, das obigen Algorithmus und noch einen anderen verwendet, den wir uns weiter unten genauer ansehen werden, mit der Kommandozeile

```
gcc -m32 -o poti.exe poti.c poti2.s
```

und starten Sie es.
Welche Zeit benötigt der oben angegebene Algorithmus auf Ihrem PC zur Berechnung des Ausdrucks (1) mit den oben angegeben Zahlen a, N und m?

(c) Der zweite in dem Programm verwendete Algorithmus ist offenbar, was die Ausführungszeit angeht, um Klassen besser. Der entsprechende C-Code dazu ist in *poti.c* auskommentiert und in *poti2.s* noch einmal etwas anders implementiert. Kommentieren Sie die einzelnen Befehle dieses Assembler-Codes. Zur Erinnerung: C legt die Parameter von rechts nach links auf den Stack und gibt das Ergebnis einer Funktion in %eax zurück.

```
#-----------------------------------------
# int poter2(int a, int N, int m)
#-----------------------------------------
poter2:
   pushl   %ebp
   movl    %esp, %ebp
   pushl   %ebx
   pushl   %ecx
   movl    $1, %ebx
   movl    8(%ebp), %ecx
loop:
   cmpl    $0, 12(%ebp)
   je      ende
   movl    12(%ebp), %eax
   andl    $1, %eax
   je      weiter
   movl    %ebx, %eax
   subl    %edx, %edx
   imull   %ecx, %eax
   idivl   16(%ebp)
   movl    %edx, %ebx
weiter:
   movl    %ecx, %eax
   subl    %edx, %edx
   imull   %ecx, %eax
   idivl   16(%ebp)
   movl    %edx, %ecx
   shrl    12(%ebp)
   jmp     loop
ende:
   movl    %ebx, %eax
   popl    %ecx
   popl    %ebx
   movl    %ebp, %esp
   popl    %ebp
   ret
```

Dazu noch eine Bemerkung: Der Befehl *imull* erwartet bei der Multiplikation von zwei positiven 32 Bit Werten, dass die eine Zahl in %eax steht und in %edx eine Null. Das Ergebnis steht dann als 64 Bit-Wert in (%edx,%eax)

idivl dividiert entsprechend die 64 Bit-Zahl in (%edx,%eax) durch den angegeben Wert. Das Ergebnis steht dann in %eax, der Rest in %edx.

(d) Multiplikationen und Division sind auch auf modernen Prozessoren immer noch recht zeitaufwändig. Wir vermuten daher, dass die Anzahl der Multiplikationen/Divisionen unsere beiden Algorithmen entscheidend beeinflusst.

Wieviele Multiplikationen/Division benötigt das erste implementierte Verfahren, um den Ausdruck (1) für eine gegebene Zahl N zu berechnen?

Beim zweiten Algorithmus lässt sich die genaue Zahl der Multiplikationen/Divisionen für ein gegebenes N nicht so einfach angeben, wohl aber eine obere Schranke. Bestimmen Sie eine solche Schranke (je kleiner, desto besser) für die Implementation im Assembler-Code. Die auskommentierte Implementation in C ist aber ähnlich schnell, auch wenn sie mehr Multiplikationen/Divisionen benötigt.

Nehmen wir an, für ein gegebenes N finden Sie für den ersten Algorithmus eine Rechenzeit von 1 Sekunde. Welche Rechenzeit erwarten Sie dann für N^2?
Wie lautet die Antwort für den zweiten Algorithmus, wenn Sie mit ihm für ein gegebenes N ebenfalls eine Rechenzeit von 1 Sekunde gebraucht haben? Hier lässt sich natürlich kein genauer Wert angeben, sondern nur die Größenordnung der Rechenzeit für N^2.