Universität des Saarlandes
FR 6.2 - Informatik
Prof. Dr. W.J. Paul
Dipl.-Ing. Christoph Baumann

Zusätzliches Übungsblatt
Informatik II
(keine Abgabe)

Hinweis: Die folgenden Aufgaben stellen eine Möglichkeit der zusätzlichen Übung zur besseren Klausurvorbereitung dar. Die Bearbeitung ist freiwillig. Das Übungsblatt wird nicht in den Übungsgruppen besprochen werden. Für Rückfragen stehen die office hours und das Forum[1] bereit. Die angegebenen Punkte dienen der Orientierung und zur Einschätzung des Arbeitsaufwands. Es können keinerlei Punkte für die Zulassung erreicht werden.

Aufgabe 1: (Codeerzeugung) **(4+4 Fleiß-Punkte)**

Sei `c` der Name einer lokalen Variable einfachen Typs, `t` ein einfacher Typ und `x` der Name einer lokalen Variable vom Typ `t*`. Generiere für die folgenden C_0-Fragmente den dazugehörenden Assemblercode:

a) `x = new t*;`

b) `return c;`

Aufgabe 2: (unsigned test code) **(4+3+8 Fleiß-Punkte)**

Angenommen wir möchten zwei Binärzahlen $a, b \in \mathbb{N}$ in Register $d^0.gpr(1)$ und $d^0.gpr(2)$ der DLX miteinander vergleichen. Das Resultat von $a < b$ soll in $R3$ gespeichert werden. Leider stellt die DLX keine unsigned Test-Operationen zur Verfügung. Deshalb muss der Vergleich zweier Binärzahlen simuliert werden.

a) Gegeben sei der folgende Code:

- `sub  3  1  2`
- `slsi 3  3` 0_{16}

Dieser funktioniert nicht für alle möglichen Zahlen. Welche Zahlen sind das? Wo liegt das Problem?

b) Angenommen man ersetzt die `sub`-Instruktion im obigen Beispiel durch eine unsigned-Subtraktions-Instruktion `subu`. Was würde sich in diesem Fall verändern? Wieso implementiert dies immer noch keinen Vergleich für Binärzahlen?

c) Der Code aus der Vorlesung löst das Problem:

- `sls  3  1  2`
- `xor  4  1  2`
- `slsi 4  4` 0_{16}
- `xor  3  3  4`

Beweise, dass nach Ausführung des Codes $\langle d^4.gpr(3)\rangle = (a < b)$ gilt!

[1] http://www-wjp.cs.uni-sb.de/forum/

Aufgabe 3: (Tests für Binärzahlen) **(2+3+3 Fleiß-Punkte)**

In dieser Aufgabe erweitern wir unseren Instruktionssatz um vier Test Operationen für Binärzahlen. Das Bit $I[4]$ wurde bisher nicht zur ALU-Berechnung gebraucht, desweiteren sei die restliche Konstruktion (bis auf ALU) dem neuen Instruktionssatz angepasst.

Damit erhalten wir den folgenden erweiterten Instruktionssatz:

$I[31:26]$	$I[5:0]$	mnemonic	Effekt
000000	001000	clr	$RD = 0^{32};$
000000	001001	sgr	$RD = (RS1 > RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001010	seq	$RD = (RS1 = RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001011	sge	$RD = (RS1 \geq RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001100	sls	$RD = (RS1 < RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001101	sne	$RD = (RS1 \neq RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001110	sle	$RD = (RS1 \leq RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	001111	set	$RD = 1^{32};$
000000	011001	ugr	$RD = (RS1 >_u RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	011011	uge	$RD = (RS1 \geq_u RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	011100	uls	$RD = (RS1 <_u RS2\ ?\ 0^{31}1\ :\ 0^{32});$
000000	011110	ule	$RD = (RS1 \leq_u RS2\ ?\ 0^{31}1\ :\ 0^{32});$

Tabelle 1: signed und unsigned Test-Operationen

Zeilen 1-8 geben die bisherigen logischen Funktionen für Two's-Complement-Zahlen die Zeilen 9-12 die neuen logischen Funktionen für Binärzahlen.

a) Wie müssen der Eingänge zur ALU verändert werden um die neuen Instruktionen zu unterstützen? Wie müssen die Eingänge belegt werden?

b) Wie berechnet sich das neg-Bit für Binärzahlen?

c) Erweitern Sie die Implementierung der ALU, so dass die Implementierung der ALU die vier neuen Instruktionen berechnen kann.

Aufgabe 4: (Typen von Ausdrücken) **(15 Fleiß-Punkte)**

In Aufgabe 2 des letzten Übungsblatts betrachteten wir Zuweisungen der Form

$$id = e$$

im Rumpf einer Funktion f. Die Typ-Kontextbedingungen erforderten eine statische Analyse von Ausdrücken, insbesondere Identifiern.

- Gib eine formale Definition der rekursiven Funktion $typ(e)$ an, die zu jedem Ausdruck e den zugehörigen Typ ermittelt!
- Argumentiere bei Identifiern über die Symboltabellen $ft(f).st$, $st.gm$ sowie die Typtabelle!
- Stelle außerdem für jeden Fall Bedingungen auf, unter denen der Typ von e wohldefiniert ist! Ansonsten soll $typ(e)$ = undefiniert zurückliefern.

Aufgabe 5: (Optimierter Aho-Ullmann-Algorithmus) **(3+2+5+2 Fleiß-Punkte)**

Der Aho-Ullmann-Algorithmus wurde in der Vorlesung basierend auf der Anzahl der Knoten eines Baumes definiert. Dies genügte, um die logarithmische Schranke für die Anzahl der benötigten Marken zu zeigen, allerdings ist dies nicht optimal.

a) Zeichne einen Graphen, für den der Aho-Ullmann-Algorithmus mehr Marken als nötig verwendet! Gib einen C_0-Ausdruck an, der der Struktur dieses Graphen entspricht! Was wäre eine bessere Richtgröße für die Auswahl des zuerst zu markierenden Teilbaums?

b) Wir betrachten Bäume mit Ingrad ≤ 2, die aus n Knoten $k_1, \ldots, k_n$ bestehen. Die Funktion $L(k)$ ordnet jedem Knoten k die Anzahl der Blätter zu, die an dem Teilbaum hängen, dessen Wurzel k ist. $S(k)$ beschreibe die Menge der direkten Söhne von k. Definiere $L(k)$ formal!

c) w bezeichnet im Folgenden die Wurzel eines Baumes mit Ingrad ≤ 2 und n Knoten. $T(w)$ sei die Anzahl von Marken, die man benötigt, um den Baum entsprechend des Markenspiels zu markieren. Beweise per Induktion über $n \geq 1$:

$$T(w) \leq \lceil \log L(w) \rceil + 1$$

d) Zeige, dass dies für $n > 1$ eine kleinere Schranke als $\lceil \log n \rceil$ darstellt!

Aufgabe 6: (Laden von Konstanten) **(3+4 Fleiß-Punkte)**

In der Vorlesung wurde gesagt, dass immediate-Konstanten $\geq 2^{15}$ wegen der sign-extension stets mit zwei Instruktionen in ein Register geladen werden müssen. Dazu gibt es eine Ausnahme.

a) Für welche immediate-Konstante $a \in \{0,1\}^{16}, \langle a \rangle \geq 2^{15}$ gilt

$$-\langle a \rangle = [a] \ ?$$

Verifiziere deine Aussage!

b) Wie kann man diese immediate-Konstante mit nur einer Instruktion in ein Register r speichern. Beweise, dass nach Ausführung der Instruktion gilt:

$$d'.gpr(r) = 0^{16} \circ a[15:0]$$

$d.gpr(0) = 0^{32}$ darf angenommen werden.

Aufgabe 7: (Codegenerierung) **(4+4+4+4 Fleiß-Punkte)**

Seien `a`, `b` und `c` lokale Variablen einfachen Typs und `nxt` eine globale Zeigervariable auf einen einfachen Typ. Generiere für die folgenden C0–Fragmente den dazugehörigen Assemblercode:

a) `a = a + 1;`

b) `nxt* = c;`

c) `if a == 0 then {a = b} else {a = - b};`

d) `while a == 1 do {a = 2};`

Aufgabe 8: (Simulationssatz) **(15 Fleiß-Punkte)**

Wir wollen den Beweis zur Korrektheit der DLX noch einmal teilweise nachvollziehen. Die folgenden Aussagen und Lemmata dürfen vorausgesetzt werden.

- Induktionsvoraussetzung: $c^i.pc = h^{2i}.pc$
- Alignment: $c^i.pc[1:0] = 00$
- Korrektheit von Incrementern
- $h^{2i+1}.IR = I(c^i), \quad ex(h^i) = (i \bmod 2), \quad reset(h^i) = 0$
- $p_h(h^{2i+1}) = p(c^i)$ für Prädikate p_h, die nur von IR abhängen
- $f_h(h^{2i+1}) = f(c^i)$ für Funktionen f_h, die nur von IR abhängen
- $A(h^{2i+1}) = c^i.gpr(RS1(c^i)), \quad co_h(h^{2i+1} = sxtimm(c^i)$
- $Aeqz(c) = (c.gpr(RS1(c)) = 0^{32}), \quad Aeqz_h(h^{2i+1}) = (A(h^{2i+1} = 0^{32})$

Abbildung zeigt die relevanten Schaltkreise der DLX-Implementierung.

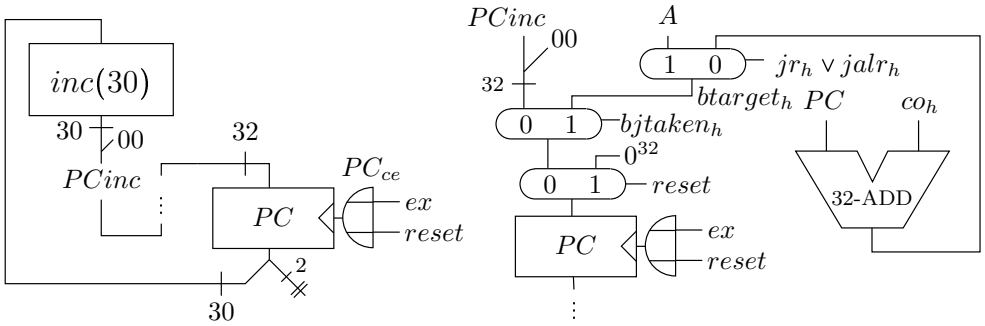

Abbildung 1: Schaltkreise zur Berechnung des nächsten pc

Die Spezifikation des pc lautet:

$$c'.pc = \begin{cases} btarget(c) & : bjtaken(c) \\ c.pc +_{32} 4_{32} & : sonst \end{cases}$$

Beweise: $c^{2i}.pc = h^{2i+2}.pc$

Zeige dazu zunächst:

- $PCinc(h^{2i+1}) = c^i.pc +_{32} 4_{32}$
- $btarget_h(h^{2i+1}) = btarget(c^i)$, wobei

$$btarget(c) = \begin{cases} c.gpr(RS1(c)) & : \quad jr(c) \vee jalr(c) \\ c.pc +_{32} sxtimm(c) & : \quad \text{sonst} \end{cases}$$

- $bjtaken_h(h^{2i+1}) = bjtaken(c^i)$, wobei
  $bjtaken(c) = btaken(c) \vee jump(c)$, $btaken(c) = branch(c) \wedge (Aeqz(c) \oplus I(c)[26])$ und
  $bjtaken_h = jump_h \vee (branch_h \wedge (Aeqz_h \oplus IR[26]))$

Aufgabe 9: (Displacement) **(5 Fleiß-Punkte)**

Das *displacement* für eine Variable $X = (lms(j), i)$ und einem Funktionsnamen f ist definiert als: $displ(f, X) = \sum_{k<i} size(ft(f).st.typ(k))$. Da für die Symboltabelle des Stackframes j gilt: $st.lms(j) = ft(f).st$ kann auch die Basisadresse von lokalen Variablen und Parametern unabhängig vom aktuellen Stackzustand berechnet werden: $ba(X) = displ(f, X)$. Zeige analog, dass ebenfalls die Basisadresse für globale Variablen nur von den C0-Parametern abhängt, nicht aber von der aktuellen Konfiguration.

Aufgabe 10: (CO-Anweisungsausführung) **(30 Fleiß-Punkte)**

Wir betrachten die folgenden beiden C_0 Programme.

```
int x;
int fak(int n) {
int x;
int y;
if n=0 then
 { x=1 }
else
 { y=fak(n-1);
   x=n*y };
return x };
int main() {
x=fak(3);
return 0 }
```

```
int x;
int main() {
int n;
x=1;
n=3;
while n>0 do
 { x=x*n;
   n=n-1 };
return 0 }
```

Zeige für beide Programme:

$$\exists t \in \mathbb{N} : va(c^t, x) = 3! \wedge c^t.pr = \varepsilon$$

Aufgabe 11: (Internal Event Signals) **(10 Fleiß-Punkte)**

In der Vorlesung wurden internen Interrupt-Signale $iev(d)[6:1]$ deklariert, die anzeigen, welcher interne Interrupt ausgelöst wird. $iev(d)$ lässt sich allein aus der DLX-Konfiguration d berechnen und hängt nicht von externen Inputs ab. Die Belegung soll lautet wie folgt:

- $iev(d)[1]$ - ill - illegal instruction
- $iev(d)[2]$ - mal - misalignment
- $iev(d)[3]$ - pff - page fault on fetch
- $iev(d)[4]$ - pfls - page fault on load/store
- $iev(d)[5]$ - trap - trap instruction
- $iev(d)[6]$ - ovf - overflow

Gib eine formale Definition von $iev(d)$ an, das heißt, definiere für jedes Bit abhängig von d, wann es gesetzt werden muss!

Aufgabe 12: (Special Purpose Register) **(10 Fleiß-Punkte)**

Beschreibe einzeln den Zweck der Special Purpose Register `SR`, `ESR`, `ECA`, `EPC`, `EDATA`, `PTO`, `PTL` und `MODE`. Zu welchem Zeitpunkt und zu welchem Zweck werden diese Register jeweils ausgelesen und beschrieben?

Aufgabe 13: (Interrupt Priority Level) **(10 Fleiß-Punkte)**

Der aktivierte Interrupt mit der höchsten Priorität wird durch die gesetzten Bits im *masked cause* festgelegt. Der Interrupt Handler muss einen Algorithmus ausführen um das gesetzte Bit mit dem kleinsten Index zu bestimmen. Schreibe ein DLX-Assemblerprogramm, das die Bits von Register $R1$ untersucht und die Nummer des kleinsten Index mit gesetztem Bit $\min\{i \in [31:0] \mid R1[i] = 1\}$ in Register $R2$ schreibt!

Aufgabe 14: (Speicher-Virtualisierung und Adressübersetzung) **(10 Fleiß-Punkte)**

In einem Betriebssystem laufen gleichzeitig verschiedene *Prozesse* auf der gleichen physikalischen Maschine. Diese Prozesse teilen sich Speicher und Prozessor.

a) Aus welchen beiden Komponenten besteht dieser Speicher? Erkläre kurz, wozu wir zwei verschiedene Komponenten benötigen.

b) Zur Virtualisierung eines großen Speichers in viele logisch separate Speicher wird für je einen Prozess eine *Page Table* benötigt.
Welche Information beinhaltet ein Eintrag in einer solchen Page-Tabelle?

c) Wie berechnet sich die *Page Table Entry Adresse*:

$ptea(c, va) =$

d) Wie berechnet sich ein *Page Table Entry*:

$pte(c, va) =$

e) Wie berechnet sich der *Physical Page Index*:

$ppx(c, va) =$

f) Wie berechnet sich die übersetzte Adresse *Physical Memory Adress* :

$pma(c, va) =$

g) Gib die Semantik der Load-Word-Instruktion an, für den Fall, dass die physikalische Maschine im *user mode* läuft und kein Page-Fault auftritt.
$c'.GPR[RD(c)] =$