Prof. Dr. Christoph Scholl
Dr. Tim Welschehold
Alexander Konrad
Niklas Wetzel

Freiburg, 18.11.2022

Betriebssysteme
Musterlösung zu Übungsblatt 5

Aufgabe 1 (3+3 Punkte)

In der Vorlesung haben Sie eine Erweiterung der Kontrolllogik der RETI kennengelernt. Diese ermöglicht zusätzlich zum Normalbetrieb auch eine Interruptbehandlung. Bisher haben Sie in der Vorlesung mehrere exemplarische boolesche Ausdrücke für Kontrollsignale passend erweitert.

Entwickeln Sie analog dazu die folgenden Kontrollsignale:

a) $\texttt{SPcken}_{\texttt{pre}}$

b) $\texttt{IVNcken}_{\texttt{pre}}$

Beachten Sie, dass sich das Signal `SPcken` nach einem Clock-Zyklus durch $\texttt{SPcken}_{\texttt{pre}}$ ergibt (für `IVNcken` entsprechend).

Lösung:

$SPcken_{pre} = [E \cdot \overline{s_1} \cdot s_0] \cdot$
$\{[\overline{I_{31}} \cdot I_{30} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}} +$
$\overline{I_{31}} \cdot \overline{I_{30}} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}} +$
$I_{31} \cdot \overline{I_{30}} \cdot \overline{I_{29}} \cdot I_{28} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}}] \cdot NB +$
$[\overline{h_2} \cdot h_1 \cdot \overline{h_0} +$
$h_2 \cdot h_1 \cdot h_0]\}$
Aktiv von P1 bis P2 (nicht einschließlich) von execute bei LOAD mit D=SP, COMPUTE mit D=SP und MOVE mit D=SP sowie hs_2 und hs_7 (bei Interruptbehandlung).

$IVNcken_{pre} = [E \cdot \overline{s_1} \cdot s_0] \cdot$
$\{[\overline{h_2} \cdot \overline{h_1} \cdot h_0] +$
$NB \cdot I_{31} \cdot I_{30} \cdot \overline{I_{26}} \cdot I_{25}\}$
Aktiv von P1 bis P2 (nicht einschließlich) von execute in hs_1 (Interrupt-Behandlung) sowie bei Befehl INT i im Normalbetrieb.

[Bei den Zeiten muss man nicht so pingelig sein, wenn es funktionieren kann! Auch Fehler in der Kodierung kann man tolerieren, wenn der richtige Befehl gemeint ist (Kommentar!)

Aufgabe 2 (2+4+2 Punkte)

Betrachten Sie folgendes Pico-C Programm mit Deklarations- und Anweisungsteil.

```
void main()
{
   //Deklarationsteil
   int x;
   int y;
   const int z = 5;

   //Anweisungsteil
   y = 3;
   x = 2;

   x = (x + ((y * z) + 10)); // vollstaendig geklammerter Ausdruck
}
```

a) Erstellen Sie die Einträge der Symboltabelle für den Deklarationsteil, nehmen Sie dabei an, dass *bds*= 128 (dezimal).

b) Geben Sie die Befehlsfolge der erweiterten ReTI an, die den im obigen Programm dargestellten vollständig geklammerten Ausdruck auswertet und das Ergebnis oben auf den Stack schreibt. Stellen Sie außerdem nach jedem Teilergebnis den aktuellen Stand des Stacks graphisch dar.

c) Angenommen Sie haben n Variablen $x_1, \ldots, x_n$, die durch (beliebige) binäre Operanden zu einem vollständig geklammerten Ausdruck verknüpft werden, sodass jede Variable exakt einmal vorkommt.

   1) Wie sieht der vollständig geklammerte Ausdruck aus, der die **maximale** Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?
   2) Wie sieht der vollständig geklammerte Ausdruck aus, der die **minimale** Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?

Lösung:

a) Symboltabelle:
st(x) = (var,int,128)
st(y) = (var,int,129)
st(z) = (const,int,'5')
0.75P für x und y, 0.5P für z

b)
```
/** Zuweisungsteil nicht benoetigt, da wir nicht so weit in der Vorlesung gekommen sind.
Muss auch nicht korrigiert werden.
/** Ab hier der eigentliche Teil der Aufgabe b), naemlich den vollstaendig
geklammerten Ausdruck auswerten. **/
/** x auf den Stack legen **/
```

```
SUBI SP 1             // Variablenbezeichner x, Adresse 128
LOAD ACC 128
STOREIN SP ACC 1      // x=2 auf Stack
/** y auf den Stack legen **/
SUBI SP 1             // Variablenbezeichner y, Adresse 129
LOAD ACC 129
STOREIN SP ACC 1      // y=3 auf Stack
/** z auf den Stack legen **/
SUBI SP 1
LOADI ACC 5           // z ist Konstante, also direkt 5 nutzen.
STOREIN SP ACC 1      // z=5 auf Stack
/** erste Klammer schliesst. y*z auswerten und Ergebnis auf Stack legen **/
LOADIN SP ACC 2          // ACC:= y=3
LOADIN SP IN2 1          // IN2:= z=5
MUL ACC IN2           // ACC:= 3*5=15
STOREIN SP ACC 2      // 15 auf den Stack (zweitoberste Stack-Zelle)
ADDI SP 1             // Stack um eins verkuerzen
/** Konstante 10 auf Stack legen **/
SUBI SP 1
LOADI ACC 10          // ACC:= 10
STOREIN SP ACC 1      // 10 auf Stack
/** zweite Klammer schliesst. 15 + 10 auswerten und auf Stack legen **/
LOADIN SP ACC 2          // ACC:= 15
LOADIN SP IN2 1          // IN2:= 10
ADD ACC IN2           // ACC:= 15+10=25
STOREIN SP ACC 2      // 25 auf den Stack (zweitoberste Stack-Zelle)
ADDI SP 1             // Stack um eins verkuerzen
/** letzte Klammer schliesst. 25+x auswerten und auf Stack legen **/
LOADIN SP ACC 2          // ACC:= 2
LOADIN SP IN2 1          // IN2:= 25
ADD ACC IN2           // ACC:= 2+25=27
STOREIN SP ACC 2      // 27 auf den Stack (zweitoberste Stack-Zelle)
ADDI SP 1             // Stack um eins verkuerzen
/** Ergebnis liegt auf Stack. Hier ist die Aufgabe erfuellt.
Ergebnis wieder in Variable x abspeichern nicht noetig fuer Aufgabenstellung. **/
```

[3P für den ReTI Code. 1P für die Stack Abbildungen.]

c) 1) $(x_1 + (x_2 + (x_3 + \ldots (x_{n-1} + x_n)) \ldots)$ mit max. n Teilergebnissen.
2) $((\ldots (x_1 + x_2) + x_3) + \ldots x_{n-1}) + x_n)$ mit max. 2 Teilergebnissen. Hier ist natürlich auch jeder andere binäre Operand außer + zulässig. [jeweils 0.5P für den Ausdruck und 0.5P für Anzahl an Teilergebnissen]

Aufgabe 3 (6 Punkte)

Die Auswertung von Vergleichsoperatoren aus Pico-C erfolgt auf der RETI, indem die Differenz der linken und rechten Seite gebildet wird und diese (je nach Vergleichsoperator) mit 0 verglichen wird.

Beispiel: $x \leq y$ wird zu $x - y \leq 0$.

Dieses Vorgehen kann jedoch zu Überläufen führen, wenn die Ausdrücke auf beiden Seiten ein unterschiedliches Vorzeichen haben.

Geben Sie zur Behandlung dieses Problems exemplarisch ein RETI-Programm an, das den logi-

schen Ausdruck $x \leq y$ korrekt zu 0 bzw. 1 auswertet, auch wenn bei der Berechnung von $x - y$ ein Überlauf auftritt.
Nehmen Sie dazu an, dass die Integer-Variable x an der Speicherstelle 10 und die Integer-Variable y an der Speicherstelle 11 abgespeichert ist.

Lösung:

```
/** x < y **/
SUBI SP 1       // Variablenbezeichner x, Adresse 10
LOAD ACC 10
STOREIN SP ACC 1 // x auf Stack
SUBI SP 1       // Variablenbezeichner y, Adresse 11
LOAD ACC 11
STOREIN SP ACC 1 // y auf Stack
LOADIN SP ACC 2  // Wert von x in ACC laden
LOADIN SP IN2 1  // Wert von y in IN2 laden
/** Zunaechst mal auf Vorzeichen testen **/
OPLUS ACC IN2   // ACC ^ IN2 (Im hoechstwertigen Bit ist eine 1, wenn das VZ unterschiedlich ist.)
AND ACC d       // Annahme: an Stelle d steht Bitmaske 10...0
JUMP= 4         // gleiche Vorzeichen -> Sprung: Standard
LOADIN SP ACC 2    // unterschiedliche Vorzeichen, Sonderbehandlung, x wieder in ACC
JUMP> 5         // x ist positiv -> Sprung: Ergebnis=0 x > y
JUMP_111 3      // x ist negativ -> unbedingter Sprung: Ergebnis=1 x < y
/***************************************/
LOADIN SP ACC 2   // Wir muessen wieder x in ACC laden, da wir den ACC zuvor ueberschrieben haben.
SUB ACC IN2     // x - y in ACC laden
JUMP<= 3        // Ergebnis 1, wenn x - y <= 0 wahr
LOADI ACC 0     // Ergebnis 0, wenn wenn x - y <= 0 falsch
JUMP 2
LOADI ACC 1     // Ergebnis 1, wenn x - y <= 0 wahr
STOREIN SP ACC 2  // Ergebnis in zweitoberste Stack-Zelle
ADDI SP 1       // Stack um eine Stelle verkuerzen
```

Abgabe: als PDF im Übungsportal bis 25.11.2022 um 12:00