Rechnerarchitektur - Übungsblatt 11
Übungsgruppe 8 - Di, 15-17 Uhr
Steffen Heil

Jan Hendrik Dithmar (Matrikelnummer 2031259)
Carsten Saar (Matrikelnummer 2030107)
Thomas Ospelt (Matrikelnummer 2029959)
Timo Adam (Matrikelnummer 2031946)

Aufgabe 1

Definiere Signale g_i und p_i, die besagen, ob die Pipeline-Stufe i ein stall-Signal generiert bzw. propagiert. Sei stall_{i+1} gegeben. Dann ergibt sich der Wert von stall_i aus folgender Tabelle

lstall_i	full_i	stall_i
0	0	0
0	1	stall_{i+1}
1	0	0
1	1	1

Wie aus der Tabelle offensichtlich hervorgeht, gelten folgende Äquivalenzen

- Die Pipeline-Stufe i generiert genau dann ein stall-Signal, wenn $\text{lstall}_i = 1$ und $\text{full}_i = 1$ gilt.
  $g_i = 1 \iff \text{lstall}_i \wedge \text{full}_i$
- Die Pipeline-Stufe i propagiert genau dann das stall-Signal der Pipeline-Stufe $(i+1)$, wenn $\text{lstall}_i = 0$ und $\text{full}_i = 1$ gilt.
  $p_i = 1 \iff \overline{\text{lstall}_i} \wedge \text{full}_i$

Definiere analog zur Konstruktion des Carry Lookahead Adders die Funktion

$$\circ : \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^2$$

$$(g_2, p_2) \circ (g_1, p_1) := (g_2 \vee (g_1 \wedge p_2), p_2 \wedge p_1)$$

welche nach dem Beweis auf Übungsblatt 8 assoziativ ist. Ebenfalls analog zur Konstruktion des CLA ergibt sich dann für die geforderte Schaltfunktion ein Schaltkreis nach Abbildung 1. Die Einhaltung der Grenzen für die Kosten und die Tiefe des Schaltkreises ergeben sich trivial, da Kosten und Tiefe des PP bekannt sind.

Aufgabe 2

Anmerkung: Es wird keine Prüfung vorgenommen, ob der Ausdruck den geforderten Eigenschaften entspricht (z.B. gültige Klammersetzung). Ist der Ausdruck nicht regelkonform, ist das Verhalten des Programms nicht definiert.

```
    R1 = 1000        // Zeiger auf die betrachtete Stelle des Ausdrucks
    R2 = 10000       // Zeiger auf das nächste freie Element im Stack
loop:                // Schleifenbeginn, nächstes Symbol ist noch zu laden
    R3 = M4(R1)      // lade erstes Wort (Symbolklasse) in R3
    R4 = M4(R1+4)    // lade zweites Wort (Wert, Zeiger oder Operator) in R4
    R1 = R1 + 8      // setze Zeiger auf nächstes Symbol
looploaded:          // Schleifenbeginn, nächstes Symbol ist bereits geladen

    R5 = (R3≤2 ? 1 : 0)                  // ist Symbol eine Konstante oder Variable ?
    PC = PC + (R5=0 ? noconstnovar : 4)  // wenn nein, auf nächste Regel testen
                                         // aktuelles Symbol ist Konstante oder Variable
    PC = PC + (R3=0 ? const : 4)         // wenn Konstante, Variablen-Auswertung überspringen
    R4 = M4(R4)                          // überschreibe Variablenadresse mit Variablenwert
const:                                   // aktuelles Symbol ist Konstante
                                         // (oder ausgewertete Variable)
    M4(R2) = R0                          // es wird eine Konstante auf den Stack gepusht,
                                         // erstes Wort 0
    M4(R2+4) = R4                        // Wert der Konstanten auf den Stack pushen
    R2 = R2 + 8                          // setze Stackpointer auf nächstes freies Element
    PC = PC + loop                       // nächstes Symbol auswerten
noconstnovar:

                                         // Symbol ist in jedem Fall ein Operator
    PC = PC + (R4≠0 ? nobracket : 4)     // wenn keine öffnende Klammer,
                                         // auf nächste Regel testen
    R5 = 2                               // Symbolklasse für Operatoren ist 2
    M4(R2) = R5                          // Symbolklasse pushen
    M4(R2+4) = R0                        // offnende Klammer auf den Stack pushen
    R2 = R2 + 8                          // setze Stackpointer auf nächstes freies Element
    PC = PC + loop                       // nächstes Symbol auswerten
nobracket:
```

```
                                        // Erinnerung: Symbol ist in jedem Fall ein Operator
    R5 = (R4=1 ? 1 : 0)                 // ist Symbol eine schließende Klammer ?
    PC = PC + (R5=0 ? nobracketconst : 4)   // wenn nein, auf nächste Regel testen
                                        // Achtung! der Fall, dass sich nicht mindestens
                                        // zwei Symbole auf dem Stack befinden, wird
                                        // vernachlässigt, da dieser Fall nur bei einem
                                        // ungültigen Ausdruck auftreten kann
    R5 = M4(R2-16)                      // lade Symbolklasse des vorletzten Symbols
                                        // auf dem Stack
    PC = PC + (R5=0 ? nobracketconst : 4)   // wenn kein Operator, auf nächste Regel testen
    R5 = M4(R2-12)                      // lade Operator-Identifikation des vorletzten
                                        // Symbols auf dem Stack
    PC = PC + (R5≠0 ? nobracketconst : 4)   // wenn keine öffnende Klammer, nächste Regel
    R5 = M4(R2-8)                       // lade Symbolklasse des letzten Symbols
                                        // auf dem Stack
    PC = PC + (R5≠0 ? nobracketconst : 4)   // wenn keine Konstante, auf nächste Regel testen
                                        // es liegt ein Ausdruck der Form „( C )“ vor
    M4(R2-16) = R5                      // kopiere letztes Symbol des Stacks an die
                                        // vorletzte Stelle, hier die Symbolklasse = 0
    R5 = M4(R2-4)                       // lade Wert des letzten Symbols des Stacks,
                                        // hier Wert der Konstanten
    M4(R2-12) = R5                      // setze Wert des vorletzten Symbols des Stacks,
                                        // Kopiervorgang beendet
    R2 = R2 - 8                         // entferne letztes Symbol von Stack,
                                        // Klammer entfernt, Konstante jetzt letzter Wert
    R5 = (R2=10008 ? 1 : 0)             // ist die Konstante auf dem Stack
                                        // das einzige Symbol ?
    PC = PC + (R5=1 ? ready : 4)        // wenn ja, Auswertung des kompletten
                                        // Ausdrucks abgeschlossen
    PC = PC + loop                      // ansonsten nächstes Symbol bearbeiten
nobracketconst:                         // es liegt keine Ausdruck der Form „( C )“ vor
```

```
                                        // Erinnerung: Symbol ist in jedem Fall ein Operator
    R5 = M4(R2-24)                      // lade Symbolklasse des vorvorletzten Symbols
                                        // auf dem Stack
    PC = PC + (R5≠0 ? noconstopconst : 4)   // wenn keine Konstante, dann Fehler
    R5 = M4(R2-16)                      // lade Symbolklasse des vorletzten Symbols
                                        // auf dem Stack
    PC = PC + (R5=0 ? noconstopconst : 4)   // wenn kein Operator, dann Fehler
    R5 = M4(R2-8)                       // lade Symbolklasse des letzten Symbols
                                        // auf dem Stack
    PC = PC + (R5≠0 ? noconstopconst : 4)   // wenn keine Konstante, dann Fehler
                                        // es gilt prio(op) = zweites Wort - 1 außer
                                        // für den Operator „-“
                                        // ⟹ die Prioritätsauswertung läßt sich
                                        // wie folgt vereinfachen:
                                        // „C1 op C2“ wird genau dann ausgewertet, wenn
                                        // - op' ≤ op (Operator-IDs werden verglichen)
                                        // - oder op' = „-“  ∧  op = „+“
    R6 = M4(R2-12)                      // lade Operator-ID des vorletzten Symbols
                                        // auf dem Stack
    R5 = (R4≤R6 ? 1 : 0)                // ist op' ≤ op ?
    PC = PC + (R5≠0 ? eval : 4)         // wenn ja, wird ausgewertet
    R5 = (R4=6 ? 1 : 0)                 // ist op' = „-“ ? (Sonderfall)
    PC = PC + (R5=0 ? noeval : 4)       // wenn nein, keine Auswertung
    R5 = (R6=5 ? 1 : 0)                 // ist op = „+“ ? (Sonderfall)
    PC = PC + (R5≠0 ? eval : 4)         // wenn ja, wird ausgewertet

noeval:                 // keine Auswertung,
                        // neuen Operator auf Stack pushen
    M4(R2) = R3         // Symbolklasse auf Stack bringen
    M4(R2+4) = R4       // Operator-ID auf Stack bringen
    R2 = R2 + 8         // setze Stackpointer auf nächstes freies Element
    PC = PC + loop      // nächstes Symbol bearbeiten
```

```
eval:                                   // Auswertung durchführen
    R5 = M4(R2-20)                      // lade Konstante C1
                                        // (vorvorletztes Symbol auf dem Stack)
    R7 = M4(R2-4)                       // lade Konstante C2
                                        // (letztes Symbol auf dem Stack)
                                        // es werden nur die Fälle op = ∨/∧/=/+/- bzw.
                                        // R6 = 2/3/4/5/6 berücksichtigt; alle anderen
                                        // Fälle dürfen an dieser Stelle nicht
                                        // auftreten, wenn der Ausdruck zulässig ist
    R6 = R6 - 2                         // vermindere op um 2, im folgenden
                                        // nur noch Dekrements und Tests auf 0 nötig
    R8 = R5 ∨ R7                        // vorsorglich Oder berechnen
    PC = PC + (R6=0 ? saveresult : 4)   // wenn wirklich Oder zu berechnen, speichern
    R6 = R6 - 1                         // prüfe auf nächsten Operator
    R8 = R5 ∧ R7                        // vorsorglich Und berechnen
    PC = PC + (R6=0 ? saveresult : 4)   // wenn wirklich Und zu berechnen, speichern
    R6 = R6 - 1                         // prüfe auf nächsten Operator
    R8 = (R5=R7 ? 1 : 0)                // vorsorglich Gleich berechnen
    PC = PC + (R6=0 ? saveresult : 4)   // wenn wirklich Gleich zu berechnen, speichern
    R6 = R6 - 1                         // prüfe auf nächsten Operator
    R8 = R5 + R7                        // vorsorglich Plus berechnen
    PC = PC + (R6=0 ? saveresult : 4)   // wenn wirklich Plus zu berechnen, speichern
    R8 = R5 - R7                        // Minus berechnen, denn nur dieser Operator
                                        // kann es an dieser Stelle noch sein
saveresult:
    R2 = R2 - 16                        // entferne die letzten zwei Symbole vom Stack
    M4(R2-4) = R8                       // ersetze Konstante (letztes Symbol auf Stack)
                                        // durch berechnete Konstante
                                        // Symbolklasse ist bereits auf Konstante gesetzt
    PC = PC + looploaded                // bereits geladenes Symbol auswerten
noconstopconst:

// wird diese Programmstelle erreicht, ist der Ausdruck ungültig
// da ein gültiger Ausdruck vorausgesetzt wurde, hier keine weitere Aktion

ready:
// Auswertung des Ausdrucks abgeschlossen
```

Aufgabe 3

Anmerkung: Die vorzeichenlose Integer-Division wird durch wiederholtes Subtrahieren des Divisors vom Dividenden erreicht, wobei die Anzahl der Subtraktionen mitgezählt wird und nur solange subtrahiert wird, bis der Dividend kleiner als der Divisor ist.

```
                                  // in R1 steht nach Voraussetzung der Dividend
                                  // und in R2 der Divisor ≠ 0
    R3 = 0                        // initialisiere den Quotienten mit 0
    R4 = R1                       // und den Rest mit dem Dividenden
    PC = PC + test                // vor Eintritt in die Schleife Bedingung prüfen
loop:
    R3 = R3 + 1                   // ansonsten Quotient inkrementieren
    R4 = R4 - R2                  // und den Rest um den Divisor vermindern
test:
    R5 = (R4<R2 ? 1 : 0)          // ist der Rest kleiner dem Divisor ?
    PC = PC + (R5=0 ? loop : 4)   // wenn nein, dann Schleife erneut durchlaufen
```

Aufgabe 4

Anmerkung: Das Kommando M_2`(writecmd+2) = R1` evaluiert unter der Voraussetzung, dass das Programm ab Speicheradresse 0 beginnt, zum Kommando M_2`(10) = R1`.

```
    R1 = 32768                    // beginne mit inkrementierter Endadresse, da zu
                                  // Beginn der Schleife sofort dekrementiert wird
loop:
    R1 = R1 - 1                   // dekrementiere Adresse
    M2(writecmd+2) = R1           // überschreibe die Immediate-Konstante des Speicher-
                                  // zugriffes bei writecmd, welche sich im hinteren
                                  // Halbwort des Instruktionswortes befindet, mit der aktuellen
                                  // Adresse, so dass beim Erreichen des Schreibkommandos bei
                                  // writecmd der Wert aus R1 an die Adresse R0+R1 = R1
                                  // geschrieben wird
writecmd:
    M1(0) = R1                    // Achtung! selbstmodifizierender Code, siehe letzte Zeile
                                  // schreibe an Adresse R1 den Wert R1 mod 2^8 (beim Byte-
                                  // Schreibzugriff werden die oberen 24 Bit abgeschnitten)
    R2 = (R1≤1024 ? 1 : 0)        // soeben schon der Anfang des Speicherblocks bearbeitet?
    PC = PC + (R2=0 ? loop : 4)   // wenn nein, Schleife ab loop erneut durchlaufen
```