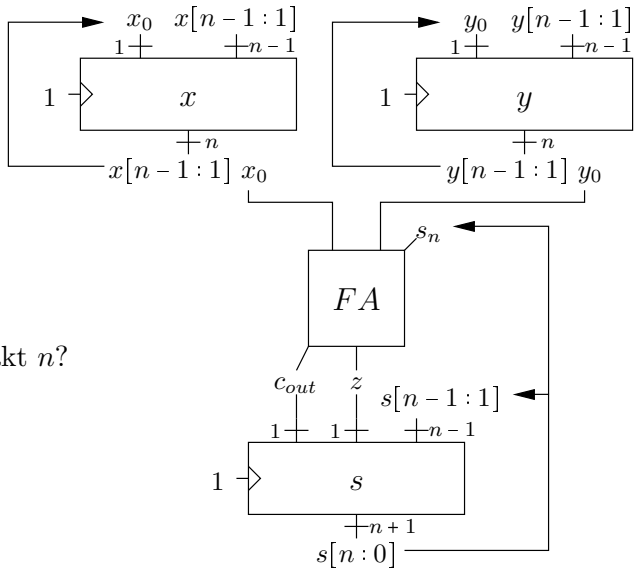
Universität des Saarlandes
FR 6.2 - Informatik
Prof. Dr. W.J. Paul
Dipl.-Ing. Christoph Baumann

8. Übungsblatt Informatik II

(Abgabe: 23.06.2008)

Aufgabe 1: (clocked circuit) (5+10 Punkte)

Gegeben sei der folgende getaktete Schaltkreis bestehend aus 2 n-Bit Registern x und y, einem $n+1$-Bit Register s sowie einem Volladdierer.



Zu Beginn stehen in den Registern x und y die Werte $x^0, y^0 \in \{0,1\}^n$. Desweiteren gilt $s_n^0 = 0$. Der Rest von Register s sei anfangs undefiniert.

a) Welchen Zweck erfüllt diese Schaltung? Spezifiziere die Signale x_0^t, y_0^t, s_n^t, z^t und c_{out}^t für den Takt t, $0 \leq t < n$!

b) Welchen Wert beinhaltet Register s im Takt n? Beweise deine Aussage!

Aufgabe 2 BONUS: (Assemblerprogrammierung) (15* Punkte)

Schreibe ein DLX-Assembler-Programm, dass alle Primzahlen $\leq n$ nacheinander in den Speicher schreibt! Dabei soll bei Wortadresse a begonnen werden und jede Zahl ein Speicherwort einnehmen. Die natürliche Zahl n stehe zu Beginn in Register 20 und die Adresse a in Register 21.
Hinweis: Implementiere das *Sieb des Eratosthenes*!

Die Lösung zu dieser Aufgabe ist elektronisch zu erstellen und per Email fristgerecht bei deinem Übungsleiter einzureichen. Unkommentierte Programme werden nicht bewertet.

Aufgabe 3: (Unvollständig geklammerte boolesche Ausdrücke) (8 Punkte)

In der Vorlesung wurde eine Grammatik für vollständig geklammerte boolesche Ausdrücke angegeben.

a) Definiere eine eindeutige Grammatik für unvollständig geklammerte boolesche Ausdrücke! Die zu verwendenden Operatoren seien $\wedge, \vee$ und $\neg$. Es gelten die bekannten Vorrangsregeln.

b) Entwickle einen Ableitungsbaum gemäß deiner Grammatik für folgenden unvollständig geklammerten booleschen Ausdruck!

$$\neg x0 \wedge x1 \vee x1 \wedge (x2 \vee x3)$$

Aufgabe 4: (C_0-Grammatik) **(7 Punkte)**

Erstelle gemäß der Grammatik von C_0 einen Ableitungsbaum mit Wurzel $\langle program \rangle$ für folgendes C_0-Programm! Es ist gestattet, Teilbäume für separate Abschnitte des Codes zu entwerfen. Die Zeilennummerierung ist kein Bestandteil des Programms.

```
 0: typedef int* intp;
 1: intp[10] a;
 2: int fak(int x)
 3: {
 4:   int y;
 5:   if x==1 then {y=x} else
 6:   {
 7:     y=fak(x-1);
 8:     y=x*y
 9:   };
10:   return y
11: };
```

```
12: int main()
13: {
14:   unsigned b;
15:   int i;
16:   b=a&;
17:   i=1;
18:   while i<11 do
19:   {
20:     b*[i-1]*=fak(i);
21:     i=i+1
22:   };
23:   return 42
24: }
```

Aufgabe 5: (syntactic sugar) **(5+5 Punkte)**

Um bequemer programmieren zu können, wollen wir die C_0-Grammatik um Strings (Zeichenketten) erweitern. Die Länge der Zeichenketten sei dabei beliebig. Die folgenden Funktionalitäten müssen von der Syntax abgedeckt werden.

- Deklarierung von String-Variablen
- vollrekursive Typdeklarationen mit Strings
- Konkatenationen von mehreren Strings bzw. Chars zu einem einzigen String mit Hilfe eines geeigneten Operators
- Zuweisung von Strings an Variablen

a) Gib die veränderten bzw. neuen Nichtterminale und Produktionsregeln an, die nötig sind, um Strings in C_0 zu integrieren! Achte darauf, dass die Eindeutigkeit der Grammatik gewahrt bleibt!

b) Schreibe C_0-Programm-Fragmente die die folgende Prozedur implementieren! Verwende dabei deine String-Syntax!

- Definiere einen Datentyp `strings`, der einen Array aus 10 Strings repräsentiert!
- Deklariere eine Variable `a` vom Typ `strings` und eine String-Variable `b`!
- Weise `b` die Konkatenation der Zeichen *s*,*s*,*0*,*8* zu! (Die Konkatenation soll programmiert werden.)
- Weise der dritten Komponente von `a` die Konkatenation des Strings *info2* mit dem Inhalt von `b` zu!

Erstelle für diese Code-Fragmente Ableitungsbäume gemäß der neuen Grammatik!