# Hinweise zur Klausur des Kurses 1704 am 12.09.1998

- Die Dauer der Klausur beträgt 3 Stunden (von 10:00 bis 13:00).
- Es sind **keine Hilfsmittel** zugelassen.
- Die Klausur umfaßt insgesamt 10 Seiten und 5 Aufgaben.
- Sie benötigen eigene Lösungsblätter.
- Den Übungsschein bzw. das Zertifikat füllen Sie bitte **vor** der Bearbeitung der Klausuraufgaben leserlich aus.
- Die Bescheinigung für das Finanzamt ist nur mit Unterschrift und Stempel gültig. Nur **vollständig ausgefüllte** Bescheinigungen werden von uns abgestempelt, unterschrieben und Ihnen zusammen mit der Klausur zugestellt.
- Schreiben Sie auf alle Lösungsblätter Ihren **Namen** und Ihre **Matrikelnummer**.
- Heften Sie bitte bei Abgabe Ihrer Arbeit die ersten Seiten des Klausurexemplares (Deckblatt, Bescheinigungen) vor Ihre Lösungsblätter. Nachträglich eingereichte Lösungen werden nicht akzeptiert.
- Die Aufgabenstellung der Klausur können Sie nach der Klausur mitnehmen.

# Aufgabe 1: Multiplizierer (30 Punkte)

Gegeben seien zwei n-Bit Binärzahlen $a = (a_{n-1}, \ldots, a_0)$ und $b = (b_{n-1}, \ldots, b_0)$, wobei $a_{n-1}$ bzw. $b_{n-1}$ das jeweils höchstwertigste Bit darstellt. Die beiden Zahlen $a, b$ sollen nach der sog. 'Schulmethode' multipliziert werden:

1. Für alle $i \in \{0, \ldots, n-1\}$ multipliziert man den Multiplikanden $a$ mit dem Bit $b_i$ des Multiplikators $b$ und schiebt das Ergebnis um $i$ Stellen nach links, was einer Multiplikation mit $2^i$ entspricht, d.h $d_i = \langle a \rangle \cdot b_i \cdot 2^i$. Hierbei ist $\langle a \rangle = \sum_{j=0}^{n-1} a_j \cdot 2^j$ die durch $a$ dargestellte Zahl.

2. Das Ergebnis $Z$ der Multiplikation erhält man durch Summation über alle $d_i$, d.h. $Z = \sum_{i=0}^{n-1} d_i$.

Im Rahmen dieser Aufgabe soll ein Multiplizierer entwickelt werden, der zwei n-Bit Zahlen nach der oben beschriebenen Methode in n Zyklen multipliziert.

1. Geben Sie für den Multiplizierer M(n,1), der n-stellige Binärzahlen mit einem einzelnen Bit $b_i$ multipliziert, den Schaltplan (auf Gatterebene) an (10 Punkte)

2. Die Teilergebnisse lassen sich auch in der folgenden Weise summieren:

$$D_0 = 0, \quad D_i = 2D_{i-1} + \langle a \rangle \cdot b_{n-i} \qquad , i = 1, \ldots, n$$

   Beweisen Sie die folgende Gleichung:

$$D_i = \langle a \rangle \cdot \sum_{j=n-i}^{n-1} b_j \cdot 2^{j-(n-i)}$$

   Aus der Gleichung folgt dann direkt, daß $D_n = \langle a \rangle \cdot \langle b \rangle$ ist. (10 Punkte)

3. Geben Sie ein Schaltwerk (auf Registerebene) für einen Multiplizierer M(n, n) an, der zwei n-stellige Binärzahlen $a, b$ nach der oben beschriebenen Methode multipliziert. Dabei stehen Ihnen die folgenden Bauteile zur Verfügung:

   - n-Bit Register, 2n-Bit Register
   - n-Bit Schieberegister
   - 2n Bit Addierer
   - M(n,1) (siehe oben)

Die Operanden $a, b$ seien dabei in dem Register A bzw. dem Schiebereigster B gespeichert. Das Ergebnis der Multiplikation soll in einem Register Z gespeichert werden. Gehen Sie davon aus, das der Inhalt des Registers Z zu Beginn 0 ist. (Hinweis: Verwenden Sie die Iteration aus Teil 2, um den Multiplizierer aufzubauen). (10 Punkte)

# Aufgabe 2: Steuerwerke (30 Punkte)

In Abbildung 1 sind die Datenpfade des ersten speicher-programmierbaren Rechners, der MARK I, vereinfacht dargestellt. Bei den Elementen ACC, IR und PC handelt es sich um 32- bzw. 13-Bit-Register, ADD ist ein 13-Bit-Addierer, SUB ein 32-Bit-Subtrahierer, Memory ein 32-Bit Speicher mit $2^{13}$ Speicherworten. Bei den trapezförmigen bauelementen handelt es sich um Multiplexer, wobei die Eingänge mit dem entsprechenden Wert des Kontrollsignals markiert sind. Das Instruktionsformat der MARK I bestand aus Opcode op (3 Bit) und Konstante co (13 Bit), wobei die restlichen 16 Bits des Instruktionswortes nicht genutzt wurden. Datenworte erstrecken sich jedoch über die gesamten 32 Bit. Folgende Befehle wurden von der MARK I unterstützt:

| op | Befehl | Funktion |
|---|---|---|
| 0 0 0 | JMP | PC ← M(co) |
| 0 0 1 | JREL | PC ← PC+M(co) |
| 0 1 0 | LOAD | ACC ← -M(co);PC ← PC+1 |
| 0 1 1 | STORE | M(co) ← ACC;PC←PC+1 |
| 1 0 0 | SUB | ACC←ACC-M(co);PC←PC+1 |
| 1 0 1 | STOP | Stop der Programmausführung |
| 1 1 x | TEST | if(ACC<0) PC←PC+2 else PC←PC+1 |

Dabei stehen ACC, PC für den Inhalt der entsprechenden Register (siehe Abb. 1), M(co) ist der Inhalt des Speichers an der Adresse co. Das Steuerwerk der MARK I bestehe aus einem Zustandsautomaten, der die folgenden Zustände umfasst:

| State | Funktion |
|---|---|
| fetch | IR=M(PC), PC=PC+1 |
| decode | |
| jump | PC=M(co) |
| jrel | PC=PC+M(co) |
| load | ACC=-M(co) |
| store | M(co)=ACC |
| sub | ACC=ACC-M(co) |
| stop | |
| test | PC=PC+1 |

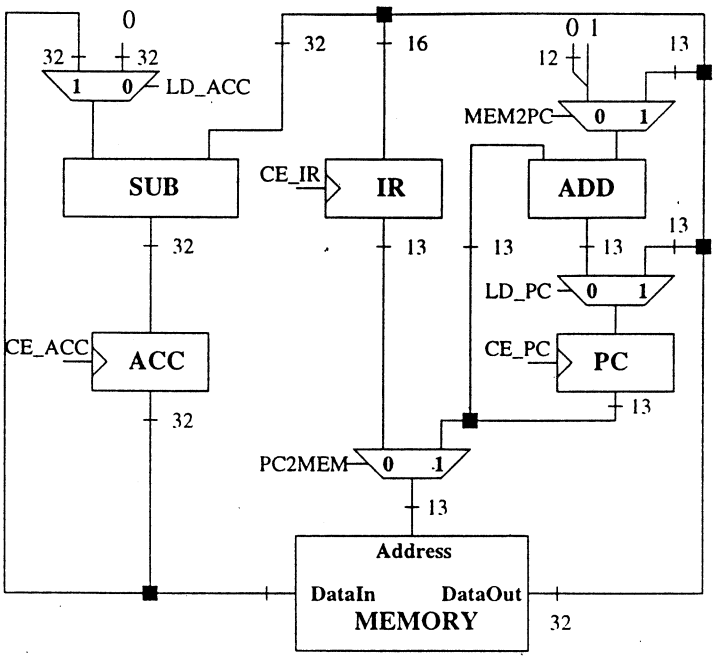


Abbildung 1: Datenpfade der MARK I (vereinfacht)

1. Entwerfen Sie für die MARK I einen Zustandsautomaten mit Hilfe der angegebenen Zustände. Geben Sie für jede Kante die Übergangsfunktion an. (5 Punkte)

2. Geben Sie für alle Zustände die jeweils aktiven (signal=1) Kontrollsignale an. Gehen Sie davon aus, das die Kontrollsignale über den gesamten Zyklus aktiv sind. (10 Punkte)

3. Erläutern Sie den Ablauf des LOAD-Befehles ausführlich (5 Punkte)!

4. Programmieren Sie die Addition auf der MARK I in Assembler. Die Operanden seien im Speicher an den Adresses 0 bzw. 1 gespeichert. Die Summe soll an Adresse 0 gespeichert werden. Kommentieren Sie Ihre Lösung ausführlich! (5 Punkte)

5. Programmieren Sie einen Links-Shift um k Stellen (Multiplikation mit $2^k$) auf der MARK I in Assembler. Die Operanden A, k liegen an den Adressen 0 bzw 1 im Speicher, Speicherzelle 2 enthält den Wert 1. Kommentieren Sie Ihre Lösung ausführlich! (5 Punkte)

## Aufgabe 3: Petri-Netze (10 Punkte)

Zeichnen Sie ein Petri-Netz für ein System, das so lange immer wieder repariert wird, bis erstmalig die Reparaturzeit den Wert T übersteigt (Danach gilt das System als ausgefallen) ! Erläutern Sie Ihre Lösung.

## Aufgabe 4: Speicherverwaltung (10 Punkte)

Beantworten Sie die folgenden Fragen :

1. Wie müßte eine Datenstruktur aussehen, die das Finden freier Hauptspeicherbereiche bei der best-fit-Strategie optimal unterstützt? (5 Punkte)

2. Wie könnte man durch Kombination von first-fit und best-fit die Einlagerung von Dateien optimieren, indem man kleinere Dateien verlagert, um für größere Platz zu schaffen ? Geben Sie ein Bild für ein Beispiel an! (5 Punkte)

# Aufgabe 5: Pipelining (20 Punkte)

Gegeben sei eine Architektur mit zwei verschiedenen Instruktionsformaten:

- ⟨OP⟩ RD, RS1, RS2 (Register-Register-Format)
- ⟨OP⟩ RD, RS1, #c(RS2) (Register-Speicher-Format)

wobei ⟨OP⟩ ein Befehl aus der Menge {ADD, SUB, AND, OR, LOAD, STORE} stammt. Bei RD, RS1 und RS2 handelt es sich um Register, c ist der Offset, der bei Speicheroperationen zum Inhalt des Registers RS2 addiert wird, um die Adresse zu erhalten. RS1 wird bei LOAD- bzw. RD bei STORE-Operationen ignoriert. Es existiert nur ein einziger Adressierungsmodus (offset c + Basis-Register RS2), Verzweigungen benutzen einen Vergleich von Registern und sind relativ zum Programmzähler. Nehmen Sie an, daß in jedem Zyklus eine neue Instruktion gestartet werden kann. Die Struktur der verwendeten Pipeline ist in Abb. 2 dargestellt, wobei

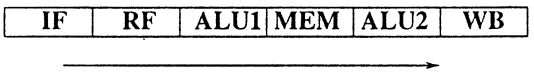


Abbildung 2: Struktur der Pipeline

**IF** Laden der auszuführenden Operation aus dem Speicher, inkrementieren des PC.

**RF** Dekodieren der auszuführenden Operation, Laden der Operanden.

**ALU1** Berechnung der effektiven Adresse bei Speicheroperationen und Verzweigungen.

**MEM** Speicheroperation ausführen.

**ALU2** Ausführung der Operation bzw. der Verzweigung

**WB** Schreiben der Ergebnisse.

1. Bestimmen Sie die Anzahl der gleichzeitig benötigten Addierer bzw. Inkrementer. Geben Sie eine Folge von Instruktionen und Pipeline-Stufen an, um Ihr Ergebnis zu begründen. (10 Punkte)

2. Bestimmen Sie die Anzahl der maximal notwendigen Lese- bzw. Schreib-Ports des Register-Files. Gehen Sie davon aus, daß der Programmzähler PC nicht Bestandteil des Register-Files ist. Begründen Sie Ihre Antwort durch die Angabe einer Folge von Instruktionen und Pipeline-Stufen. (10 Punkte)