FernUniversität
Gesamthochschule in Hagen

Fachbereich Informatik

Lehrgebiet Technische Informatik 11
Prof. Dr. Jörg Keller

Fernuniversität - Gesamthochschule D-58084 Hagen

Informatikzentrum
Feithstr. 142
D-58084 Hagen

Vorname, Name _____

Strae, Hausnr. _____

PLZ, Ort. _____

Klausur zum Kurs

Einführung in die Rechnerarchitektur (1704)

am 0508.2000

Klausurort:	
Vorname, Name:	
Matrikelnummer:	

Aufgabe	1	2	3	4	5	6	Summe
Punkte	20	20	20	10	10	20	100
bearbeitet							
erreichte Punkte							

Aufgabe 1: Addierer (20 Punkte)

Betrachten Sie den in Abbildung 1 dargestellten rekursiven Aufbau eines n-Bit (n Zweierpotenz) Conditional-Sum-Addierers (CSA) zur Addition zweier Binärzahlen

$$a = (a_{n-1}, \ldots, a_0), \qquad b = (b_{n-1}, \ldots, b_0)$$

Bei diesem Addierer werden die niederwertigen Teile

$$a_l = (a_{n/2-}, \ldots, a_0), \quad b_l = (b_{n/2-}, \ldots, b_0)$$

der beiden Binärzahlen in einem $n/2$-Bit CSA addiert, während die höherwertigen Teile

$$a_h = (a_{n-1}, \ldots, a_{n/2}), \qquad b_h = (b_{n-1}, \ldots, b_{n/2})$$

in zwei weiteren $n/2$-Bit CSA sowohl mit Eingangs-Übertrag 0 als auch mit Eingangs-Übertrag 1 addiert werden. Das korrekte höherwertige Teil s_h des Ergebnis s

$$s = (s_n, s_{n-1}, \ldots, s_0)$$

wird anschließend mit Hilfe eines Multiplexers (MUX) auf Basis des Übertrags des niederwertigen Ergebnis s_l selektiert. Ein l-Bit CSA entspricht dem bekannten Volladdierer (VA).

1. Beweisen Sie die Korrektheit des Conditional-Sum-Addierers, d.h. daß das Ergebnis s als Binärzahl der Summe der beiden Binärzahlen a und b entspricht. Benutzen Sie dabei die Methode der vollständigen Induktion. (10 Punkte).

2. Geben Sie anhand von Abbildung 1 eine rekursive Formel für die Kosten C (Anzahl der Gatter) und Tiefe T (Anzahl der Gatter auf dem längsten Pfad durch das Schaltnetz) eines n-Bit Conditional-Sum-Addierers an. Gehen Sie dabei davon aus, daß alle Gatter Kosten und Tiefe 1 besitzen, d.h. $C(\mathrm{MUX}_n) = 3n + 1$, $T(\mathrm{MUX}_n) = 3$, C(VA) = 5, $T(\mathrm{VA}) = 3$ ist. (5 Punkte)

3. Benutzen Sie das folgende Lemma:

   Sei $f : \mathrm{N} \to \mathrm{N}$ eine Funktion mit $f(1) = c$ und $f(n) = a \cdot f(n/b) + g(n)$ für alle Potenzen $n = b^k$ von b. Dann gilt

   $$f(n) = c \cdot n^{\log_b a} + \sum_{i=0}^{\log_b(n)-1} a^i \cdot g(n/b^i)$$

   für alle Potenzen n von b.

   um auf Basis Ihrer Ergebnisse aus 2) eine geschlossene Form für Kosten und Tiefe eines n-Bit Conditional-Sum-Addierers anzugeben. (5 Punkte)

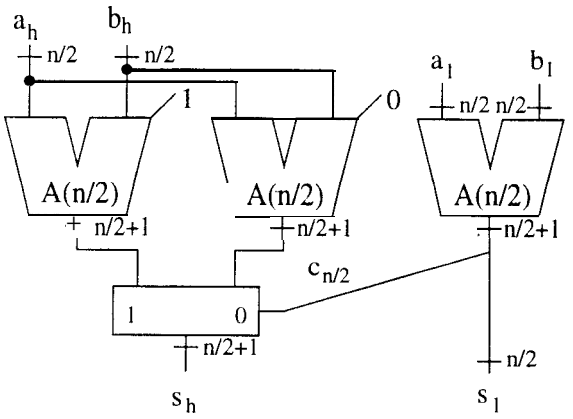

Abbildung 1: Conditional-Sum-Addierer (CSA)

Aufgabe 2: Assembler-Programmierung (20 Punkte)

In Abbildung 2 sind die Datenpfade der MARK 1, des ersten speicher-programmierbaren Rechners, 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 (LD_ACC, LD-PC, PC2MEM, MEM2PC) markiert sind. Die restlichen Kontrollsignale steuern das Laden der einzelnen Register (CE-ACC, CE-IR, CE-PC: bei 1 wird das Register geladen) bzw. die Art des Speicherzugriffs (MW: 0 : Lesen, 1 : Schreiben). Das Instruktionsformat der MARK 1 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 1 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, M(co) ist der Inhalt des Speichers an der Adresse CO.

1. Programmieren Sie die Addition auf der MARK 1 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, indem Sie zunächst die Vorgehensweise beschreiben und anschließend das kommentierte Assembler-Programm angeben. (4 Punkte)

2. Programmieren Sie einen Links-Shift um k Stellen (Multiplikation mit 2^k) auf der MARK 1 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, indem Sie zunächst die Vorgehensweise beschreiben und anschließend das kommentierte Assembler-Programm angeben. (8 Punkte).

3. Betrachten Sie den Einsatz von Arrays der Länge n auf der MARK 1. Die Länge *n* soll dabei z.B. an Addresse 0 im Speicher liegen. Warum ist der Zugriff auf die einzelnen Elemente des Arrays mit den vorgestellten Befehlen nicht möglich ? Schlagen Sie eine Änderung des Befehlssatzes vor und geben Sie ggf. notwendige Modifikationen an den Datenpfade der MARK 1 an. (8 Punkte)

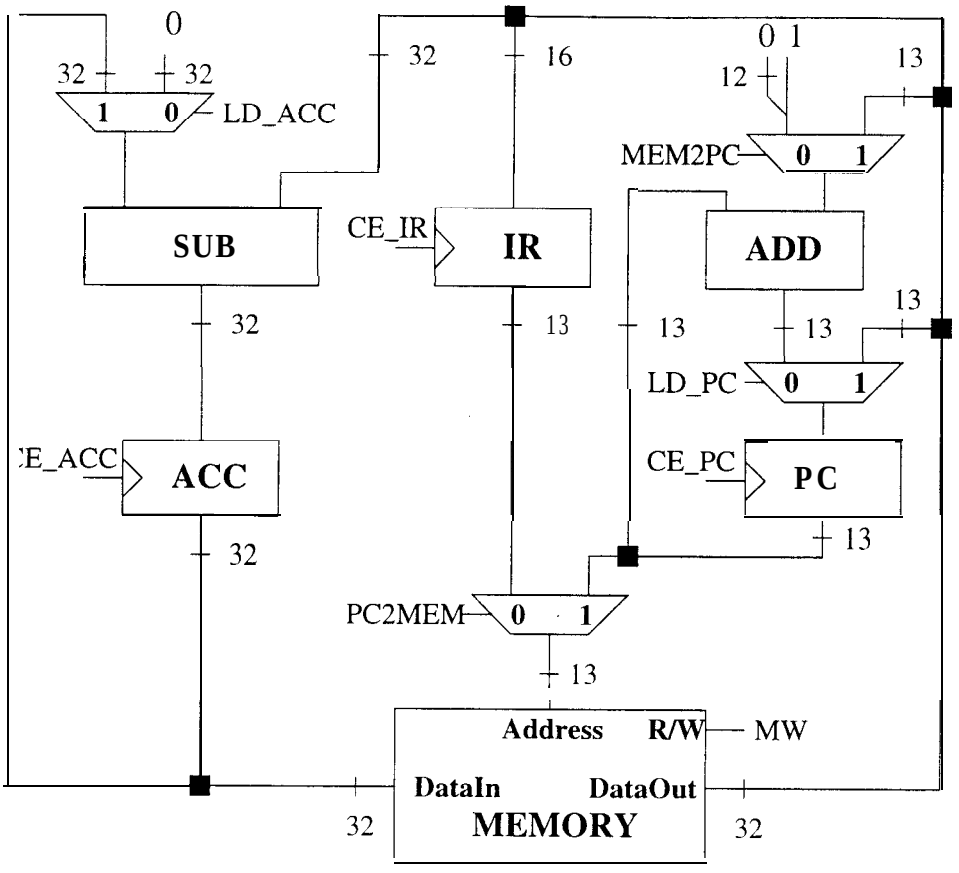

Abbildung 2: Datenpfade der MARK 1 (vereinfacht)

Aufgabe 3: Zustands-Automaten (20 Punkte)

Das Steuerwerk der MARK 1 (siehe Aufgabe 2) 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

- Entwerfen Sie für die MARK 1 einen Zustandsautomaten mit Hilfe der angegebenen Zustände. Geben Sie für jede Kante die Übergangsfunktion an. (5 Punkte)
- 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. (5 Punkte)
- Implernentieren Sie den Zustandsautomaten der MARK I als mikroprogrammierte Steuerung. Kodieren Sie dabei das Steuerwort wie folgt :

  CE_ACC, LD_ACC, CE-IR, LD_PC, CE-PC, PC2MEM, MEM2PC, MW.

  Geben Sie sowohl die Datenpfade der Steuerung als auch den Inhalt des Mikroprogramm-Speichers an und kommentieren Sie Ihre Lösung ausführlich. (10 Punkte)

Aufgabe 4: Caches (10 Punkte)

Gegeben sei ein 8 Kilobyte großer Cache mit 1024 Regionen (Cache-Lines). Die Organisation des Caches ist in den einzelnen Aufgaben spezifiziert. Zu Beginn soll das Cache jeweils leer sein.

1. Wieviele Bytes enthält eine einzelne Region ? (2 Punkte)

2. Das obige Cache sei voll-assoziativ mit einer Zugriffszeit von 2 (hit) bzw. 10 (miss) Takten, wobei misses mit einer Wahrscheinlichkeit von 2 % auftreten. Berechnen Sie aus diesen Werten den Erwartungswert der Speicherzugriffszeit. (3 Punkte)

3. Bei einem direct-mapped Cache werden nacheinander lesende Zugriffe auf die Bytes an den Adressen 3, 8199, 8200, 5, 8205 (dezimal) durchgeführt. Falls das gewünschte Byte nicht im Cache vorhanden ist (Cache-Miss), wird die entsprechende Region komplett aus dem Hauptspeicher geladen. Bei welchen Zugriffen tritt unter diesen Voraussetzungen ein Cache-Miss auf ? (5 Punkte)

Aufgabe 5: Parallel-Rechner (10 Punkte)

Gegeben sei ein Parallel-Rechner mit n Prozessoren. Für ein gegebenes Problem existiert ein paralleles Programm, das mit n Prozessoren $T(n)$ bzw. mit 1 Prozessor $T(1)$ Zeiteinheiten für die Lösung benötigt. Um den Speedup $S(n)$ und die Effizienz E(n) des parallelen Programms zu berechnen, benutzt man folgende Formeln:

$$S(n) = \frac{T(1)}{T(n)} \qquad E(n) = \frac{S(n)}{n}$$

1. Angenommen, das parallele Programm besitzt einen inherent sequentiellen Anteil c, d.h. nur $cT(1)$ Zeiteinheiten arbeiten alle n Prozessoren parallel, die restlichen $(1-c)T(1)$ Zeiteinheiten arbeitet nur ein einzelner Prozessor. Berechnen Sie unter diesen Voraussetzungen Speedup und Effizienz. (5 Punkte)

2. Das Ergebnis aus Teil 1) nennt man auch Amdahl's Gesetz. Welche Konsequenzen hat dieses Gesetz für den sequentiellen Anteil des Programms, wenn man z.B. mit 100 Prozessoren einen Speedup von 80 erreichen will ? (5 Punkte)

Aufgabe 6: Pipelining (20 Punkte)

Betrachten Sie den in der umseitigen Tabelle dargestellten Instruktions-Satz einer RISC-Architektur mit 16 Registern (RI bis R16). Eine Implementierung des Instruktionssatzes verwendet eine 5-stufige Pipeline, wobei die einzelnen Stufen wie folgt aussehen:

IF Laden der auszuführenden Instruktion. inkrementieren des PC.

RF Dekodieren der auszuführenden Instruktion, Laden der Operanden.

EXE Ausführung der Operation bzw. Berechnung der effektiven Adresse

MEM Speicheroperation ausführen.

WB Schreiben der Ergebnisse in den Registersatz.

Gehen Sie davon aus, daß in jedem Takt ein neuer Befehl gestartet werden kann und jeder Befehl genau einen Takt pro Pipeline-Stufe benötigt. Ferner arbeitet die Pipeline ohne Interlocks, d.h. Hazards müssen bei der Programmierung entsprechend berücksichtigt werden. Betrachten Sie das folgende Assembler-Programm:

```
ADD     R1, R2, R3
SUB     R4, R5, R1
AND     R6, R1, R7
OR      R8, R1, R9
XOR     R10, R1, R11
```

1. Stellen Sie die Abarbeitung des Programms in der Pipeline ausführlich dar. Wie viele Takte dauert die Abarbeitung des Assembler-Programms ? Welche Probleme treten bei der Abarbeitung auf ? (10 Punkte)

2. Modifizieren Sie das Assembler-Programm aus Teil 1 entsprechend, um die obigen Probleme zu beheben. Wie viele Takte dauert die Abarbeitung der neuen Version ? (5 Punkte)

3. Zur Leistungssteigerung wird in Pipelines oft das sog. Result-Forwarding (Bypass) benutzt, d.h. es werden zusätzliche Datenpfade eingebaut, z.B. vom Ausgang der EXE-Stufe zum Eingang der EXE-Stufe (EXE-EXE-Bypass), vom Ausgang der MEM-Stufe zum Eingang der EXE-Stufe (MEM-EXE-Bypass) und vom Ausgang der WB-Stufe zum Eingang der EXE-Stufe (WB-EXE-Bypass). Wieviele Takte werden unter diesen Voraussetzungen für die Abarbeitung des Assembler-Programms benötigt ? Treten die oben erwähnten Probleme weiterhin auf ? (5 Punkte)

Befehl	Parameter	Beschreibung
li	rl, c	Das Register r1 erhält den Wert c.
Id	rl, c, r2	Lädt den Inhalt des Speichers an Adresse r2 + c in Register rl .
st	rl, c, r2	Speichert den Inhalt des Registers rl an der Adresse r2 + c im Speicher.
add	r 1,r2,r3	Addiert den Inhalt von Register r2 und Register r3 und speichert das Ergebnis in Register r1 ab.
sub	r 1,r2,r3	Subtrahiert den Inhalt von Register r3 vom Inhalt des Registers r2 und speichert das Ergebnis in Register R1 ab.
and	r 1,r2,r3	Verknüpft die Inhalte der Register r2 und r3 durch bitweises UND und speichert das Ergebnis in Register r1 ab.
or	r1,r2,r3	Verknüpft die Inhalte der Register r2 und r3 durch bitweises ODER und speichert das Ergebnis in Register r1 ab.
xor	r 1,r2,r3	Verknüpft die Inhalte der Register r2 und r3 durch bitweises EXKLUSIV-ODER und speichert das Ergebnis in Register rl ab.
sgr	r 1,r2,r3	Vergleicht den Inhalt der beiden Register r2 und r3. Ist r2 grer als r3, wird in Register rl der Wert 1 gespeichert, ansonsten 0.
seq	r1,r2,r3	Vergleicht den Inhalt der beiden Register r2 und r3. Ist RS 1 gleich RS2, wird in Register r1 der Wert 1 gespeichert, ansonsten 0.
sls	r 1,r2,r3	Vergleicht den Inhalt der beiden Register r2 und r3. Ist RS 1 kleiner als RS2, wird in Register rl der Wert 1 gespeichert, ansonsten 0.
beq	rl, c	Vergleicht den Inhalt von Register rl mit 0. Ist der Inhalt gleich 0, wird ein Sprung um c Befehle durchgefhrt. Ansonsten wird der nachfolgende Befehl ausgefhrt.
jmp	c	Führt einen unbedingten Sprung um c Befehle aus.
nop		Dieser Befehl liefert kein Ergebnis.