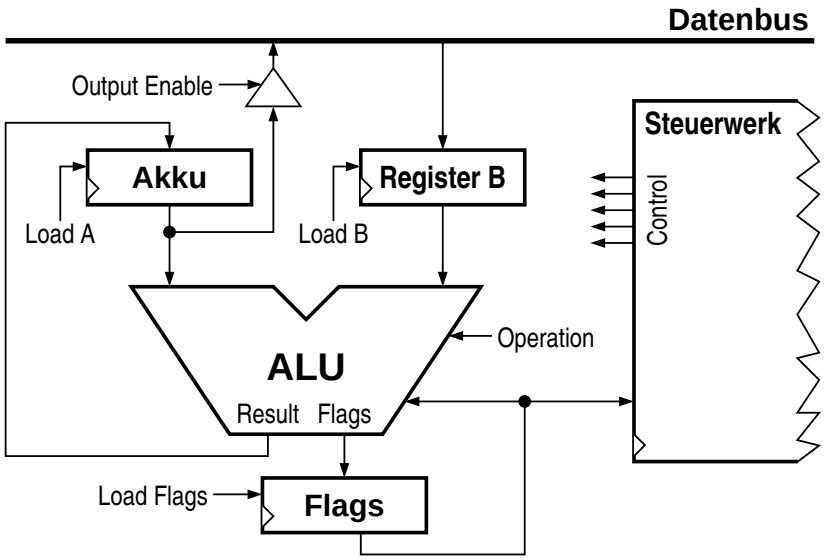
TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Technische Informatik, Professur für Prozessorentwurf

Rechnerarchitektur: Übungssatz 11

Aufgabe 11.1

Gegeben sei das folgende einfache Rechenwerk mit Akkumulator:



(a) Wie viele Operandenadressen enthält ein typischer arithmetischer Befehl einer solchen Akkumulatorarchitektur?

(b) Auf die Angabe welcher Adressen kann durch welche Adressierungsmodi verzichtet werden?

(c) Beschreiben Sie den zeitlichen Ablauf der Ausführung des Befehls `ADD [0x200]` *im Rechenwerk*! Der Speicher führt gerade den Speicherzugriff aus, der Speicheroperand liegt auf dem Datenbus. Welche Operation führt die ALU aus?

(d) Wie sieht im Vergleich dazu der Ablauf für den Befehl `LOAD [0x240]` aus?

(e) Wozu verwendet das Steuerwerk die im Rechenwerk gebildeten Flags?

(f) Welche Datenverarbeitungsbefehle der ALU könnten die Flags selbst weiterverarbeiten?

(g) Welche Befehlsklassen verändern die Flags üblicherweise nicht?

Aufgabe 11.2

Gegeben sei eine 8-Bit-Akkumulator-Architektur, die über einen byteweise adressierten Speicher mit 10 Bit Adressraum verfügt. Das Rechnwerk arbeitet im 2er-Komplement und bildet die üblichen Flags. Datenworte, die sich über mehrere Speicherzellen erstrecken, werden in Big Endian abgelegt.

Es sind die folgenden Befehle bekannt:

Mnem.	Maschinencode	Beschreibung	Aktualisierte Flags
`JPf L`	`00<f:2> <O:4>`	Bedingter Sprung bei gesetztem Bedingungsflag.	—
`JNf L`	`01<f:2> <O:4>`	Bedingter Sprung bei nicht gesetztem Bedingungsflag.	—
`JMP L`	`1000 00<O.hi:2>, <O.lo:8>`	Unbedingter Sprung.	—

- Die Assemblermnemoniken verwenden benannte Sprungmarken (Label).
- Die Sprungziele sind PC-relativ und bestimmen sich aus dem im 2er-Komplement vorliegenden, vorzeichenbehafteten Sprungoffset O relativ zur Adresse des regulären Folgebefehls.
- Das den Sprung bedingende Flag ist wie folgt kodiert:

Z	00	–	Zero	Null als Ergebnis.
C	01	–	Carry	Auslaufender Übertrag.
V	10	–	Overflow	Arithmetischer Überlauf (vorzeichenbehaftet).
S	11	–	Sign	Negatives Ergebnis.

`NOT`	`1000 0100`	A←$\bar{A}$	Invertiere Akkumulator bitweise.	Z, S
`INC`	`1000 0110`	A←A+1	Inkrementiere Akkumulator.	Z, C, V, S
`LD [P]`	`1100 00<P.hi:2>, <P.lo:8>`	A←<P>	Lade Akkumulator mit Speicherwert an Adresse P.	—
`ST [P]`	`1100 01<P.hi:2>, <P.lo:8>`	<P>←A	Kopiere Akkumulator an Speicherplatz mit Adresse P.	—
`ADD [P]`	`1100 10<P.hi:2>, <P.lo:8>`	A←A+<P>	Addiere Speicherwert an Adresse P zum Akkumulator.	Z, C, V, S
`ADC [P]`	`1100 11<P.hi:2>, <P.lo:8>`	A←A+<P>+C	Addiere Speicherwert und Carry-Flag zum Akkumulator.	Z, C, V, S

(a) Wie groß ist der maximal adressierbare Speicherbereich?

(b) Disassemblieren Sie den Maschinencodebefehl 00_{16}! Welche Auswirkungen hat er? Welche alternative Mnemonik bietet sich an?

(c) Bestimmen Sie die Ergebnisse und die Flags nach der Addition folgender Wertpaare mit Hilfe des `ADD`-Befehls! Wann sind Ergebnisse in vorzeichenloser oder in vorzeichenbehafteter Interpretation von einer Bereichsüberschreitung betroffen?

i. $27_{16} + 6C_{16}$,

ii. $E6_{16} + 1C_{16}$,

iii. $43_{16} + 84_{16}$ und

iv. $9A_{16} + 66_{16}$.

(d) Die 16-Bit-Variablen a und b liegen ab den Adressen `0x200` bzw. `0x202` im Speicher. Schreiben Sie ein Programm, das $c = a + b$ berechnet und das Ergebnis c im Speicher ab Adresse `0x204` ablegt! Geben Sie sowohl die Befehlsmnemoniken als auch den hexadezimalen Maschinencode an und kommentieren Sie die Rechenschritte!

(e) Wie könnte der Befehl `ADC [0x202]` durch die anderen simuliert werden, wenn er nicht zur Verfügung stünde?

(f) Erweitern Sie die Lösung aus (d) so, dass $c = |a + b|$ berechnet wird!