PROTOKOLL

Stand: 20.06.2003

Farberklärung:

Xxx ➔ Rico

Xxx ➔ Christoph

Xxx ➔ Sascha

Xxx ➔Vitali

1. CPU-Architektur:

1.1 Busse/Speicher

- Datenbus CPU <==> ROM 8 Bit
- Adressierung des Programmspeichers über 9 Bit (8bit Adresse + ein CS-Signal ➔ Programm in 2 ROMs verteilt ➔ damit 512 Zeilen Code möglich, Befehlswortbreite 8 Bit - siehe Befehlssatz)
- Verwendung von einem EEPROM für Flaschendaten (4x4-Bit Matrix mit 8-Bit-Feldern die Flasche beschreiben ➔ 8-Bit-Datenbus) – siehe DIAGRAMM
- Daten werden aus Schnittstelle kommend in 8-Bit-puffer gespeichert, um sie als komplettes Wort auf den Bus legen zu können (s. Buswandler)
- 4 Register (davon 3 „general purpose register“ mit 8 Bit und ein „special purpose register“ mit 16 Bit) zzgl. PC, IR, Zustandsspeicher
- ein GPR wird für Speicherung von Flags verbraten (z.Bsp.: 1 Bit Karte vorhanden, falls Karte im laufenden Betrieb eingesteckt wurde)

Nutzung der Register:

- SPR D ist Summenregister und NUR dafür zu benutzen
- A ist Dateneingangsregister (hier landet temporär die Kanalnummer)
- B ist Dateneingangsregister (für die Daten des Kanals)
- C ist FlagRegister und NUR zum setzen / abfragen von Flags zu benutzen

Setzen eines Flags:

- OR A C (wobei in A 0010.0000 steht, wenn dass dritte Bit gesetzt werden soll)

Abfragen eines Flags:

- OR C A (wobei in A 1101.111 steht, wenn dass dritte Bit abgefragt werden soll werden soll)
- ADD B A (wobei in B 0000.0000 steht)
- Jump_O B (wobei in B der Offset für den Jump steht)

1.2 Alu:

Addierer:
2 x 8 Bit auf 8 Bit + Carry-Flag ➔ falls Carry = 1 & erstes Bit der zu addierenden Zahl = 0 (also pos. Zahl) ➔ unser "Schattenregister" für die höherwertigen 8 Bit inkrementiert dann automatisch während das Register für die unteren 8 Bit das Ergebnis der ALU speichert. Falls Carry = 1 & erstes Bit der zu addierenden Zahl ist 1 (also neg. Zahl) ➔ dekrementiere Schattenregister.

Hierzu hat Sascha noch den Ablauf einer positiven und negativen Addition zum in das Dokument einfügen.

Die Alu muss ihr Ergebnis in einem internen Register speichern können, damit bei der Befehlsrealisierung das Ergebnis im nächsten Takt geholt werden kann, was wiederum sein muß, da wir nur zwei Busse haben ...
Wir haben 3 Busse!! (Sh. Diagramm ganz unten!)

ALU-Operationen: ADD, OR (für Flag-Register), INC, DEC

EINGÄNGE	**AUSGÄNGE**
2 x Daten 8 bit (A-Bus, B-Bus)	1 x Daten 8 bit
2-Bit-Steuervektor (ADD, INC, DEC, OR) 00 ➔ ADD 01 ➔ INC 10 ➔ DEC 11 ➔ OR	Carry (in CPU direkt zu Flag-FlipFlop)
	IsZero (falls Ausg. ‚0000 0000') (in CPU direkt zu Flag-FlipFlop)
	IsOne (falls Ausg. ‚1111 1111') (in CPU direkt zu Flag-FlipFlop)

1.3 Befehlssatz

Befehle sind jeweils 8-Bit lang (4Bit Opcode + 2x2 Bit Operanden)
➔ wir bauen eine zwei-Adress-CPU

Variable Dauer der Befehlsausführung
(z.B. ein 16-Bit-ADD dauert einfach länger als ein 8-Bit-ADD)
Steuerwerk ist endlicher Zustandsautomat, welcher die Befehlsausführung in „hartkodierter" Art und Weise mit Sequentierer ausführt.

Wir erlauben register-direkt und register-indirekt als Adressierungsmodi.
Alle nicht anders benannten Adress-Modi sind direkt.

Es gibt einen Store-Befehl mit einem Operand (Register) ➔ anhand des adressierten Registers wird entschieden, ob 8 oder 16 Bit versandt werden.
Der Store-Befehl soll den Inhalt eines Registers über die Schnittstelle transportieren.

Arithmetik	
Add Quelle,Ziel	falls Ziel 16Bit Register, dann Addwide (d.h. Addition inkl. Auswertung des Carry-Flags)
OR Quelle, Ziel	bitweises ODER der Operanden in den Registern
INC Quelle, Ziel	Inkrementieren des Registerinhaltes und schreiben nach Ziel
DEC Quelle, Ziel	Dekrementieren des Registerinhaltes und schreiben nach Ziel
Transport	
Load Ziel1, Ziel2	Ziel1 ist Kanalnr. Ziel2 ist Register für Daten auf Kanal
LoadOp #DO,Ziel	Direktop. DO nach Ziel (Register) laden (DO folgt der Anweisung als eigenes Datenwort.)
Store Quelle @Ziel besser Store Daten @Kanal	Wie Load nur andersherum Quelle 16 Bit ➔ anderer Befehl: erst die unteren 8 Bit senden; dann Wartezustand bis I/O-Wandler „Ready" sagt, dann die oberen 8 Bit + Wartezustand; dann normal weiter ...
Move @Quelle Ziel	Kopie aus Rom in Register, daher Quelle indirekt
Copy Quelle Ziel	Kopie von Quelle nach Ziel (Register – Register – Transfer)
Sprung	
Goto Register	Unbedingter Sprung relativ zum PC um „lange Sprünge" durch den ganzen ROM mittels mehreren GOTOs realisieren zu können
JMP_E Reg #Offset	jump if equal to zero then GOTO [PC]+Offset [geprüft wird nicht mehr der letzte arithmetische ausdruck, sonndern die Zahl die in Register steht – brauchen Sascha und Chr im Steuerwerk] wofür braucht ihr das? wie soll das dann funktionieren? Wer testet das Register??
JMP_G Reg #Offset	jump if greater than zero then GOTO [PC]+Offset
JMP_O Reg #Offset	jump if equalt to "one" then GOTO [PC]+Offset
#	➔ bedeutet Direktoperand
@	➔ bedeutet Register indirekt

1.4 „4-auf-8-Bit-Bus-Wandler“:

Allgemein:

- Wandler ist Zustandsautomat
- Liest anliegende Adresse von Kommunikationsschnittstelle ab
- die Kanalnummer (Adresse) der ankommenden Daten bestimmt die festgelegte Reaktion des Wandlers (z.B. Flaschecode kommt auf Kanal xy ➔ 8 Bit [2x4 Bit] sind zu lesen, dann Ready-Flag setzen)
- beim Schreiben ist die Adresse anzulegen - aus dem Kanal folgt wieder feste Reaktion [entweder 4 Bit (niederwertigste) oder 2x4 Bit]

Lesevorgang - Funktionsweise der Schnittstelle:
Solange keine Daten zum Lesen vorhanden sind liegt als Kanalnummer ‚0000‘ an der Schnittstelle an. Falls Daten auf einem Kanal anliegen, wird diese Kanalnummer auf den Ausgang der Schnittstelle gelegt. Die Daten (wie auch die Kanalnummer) liegen solange an, bis sie gelesen wurden.
Dann wird die nächste Kanalnummer, dessen Daten angekommen sind, an die Schnittstelle gelegt bzw. „0000“ falls keine weiteren Daten vorhanden sind.

Schreibvorgang - Funktionsweise der Schnittstelle:
Zum Schreiben werden die Eingangssignale entsprechend gesetzt, dann die Kanalnummer geschrieben, dann auf „Daten“ umgeschaltet und die Daten geschrieben. Der Schreibvorgang ist beendet, wenn das entsprechende Signal anliegt.

Eingänge

1 Bit ➔ in-out	< CPU
1 Bit ➔chip-select	< CPU
1 Bit ➔ senden beendet	< IO-Schnittstelle

Ausgänge

1 Bit ➔ ready-flag	> CPU
1 Bit ➔ Auswahl	> IO-S.
1 Bit ➔ Richtung	> IO-S.
1 Bit ➔ Adresse	> IO-S.

Bidirektional (Busse):

8 Bit ➔ Daten	<> CPU (Datenbus)
4 Bit ➔ Adresse	<> CPU (Datenbus)
4 Bit ➔ Daten	<> IO-Schnittstelle

2.Externe Kanäle

Kanalnummer	Angeschlossene Einheit	Datenformat	Datenflußrichtung
1	Kartenleser	In: 4 Bit Out: 16 Bit unsigned int	bidirektional
2	Drucker	In: 4 Bit Out: 16 Bit unsigned int	bidirektional
3	Externe Einheit	In: 4 Bit Out: 16 Bit unsigned int	bidirektional
4	Flaschensensor	ersten 4 Bit unsigned int	in
5	Flaschensensor	zweiten 4 Bit unsigned int	in
6	Interrupt- und Korrektureinheit	4 Bit signed int (*)	in
7	Kunden- / Servicekommunikation (z.Bsp.: Fehlerausgabe)	4 Bit unsigned int	Bidirektional Wieso ???
8	Fehler- / Zustandserkennungseinheit	4 Bit unsigned int	in
9 - 15	noch unbelegt		

3. Festlegungen zur Steuereinheit

- es wird prinzipiell versucht bei einer zwei-Operanden-Operation die Quelle (1. Op.) auf BUS A und das Ziel (2. Op.) auf BUS B zu legen. [Ziel = Ziel op Quelle]

4. Allgemeine Infos

max. Flaschenanzahl bzw. Kastenanzahl ist: 256, da 8 Bit

Flaschenerkennungseinheit liefert 8-Bit Code der erkannten Flasche bzw. des erkannten Kasten (2x4 Bit sequentiell geliefert auf einem Kanal), Endeflag: , 1111 1111' wird geliefert von Sensor, wenn Benutzer ‚Fertig'-Knopf gedrückt oder Zeit abgelaufen.

Pfandsummenausgabe 16 Bit (4x4 Bit sequentiell geliefert auf einem Kanal) entweder an Drucker, an Chipkarteneinheit oder an externe Schnittstelle (z.B. für Monitoring) ➔ insgesamt also 3 Kanäle. Alle Einheiten senden Bestätigung ob die Daten korrekt empfangen oder nicht. Das Senden kann nur entweder an Drucker oder an Chipkarteneinheit gesendet werden (externe Einheit geht immer!).

Chipkarteneinheit ist gleichzeitig Sender und Empfänger (sendet Info, ob eine Karte drin ist oder nicht bzw. Daten erfolgreich empfangen)

Defekte Karten werden als "nicht vorhanden" gewertet.

Kommunikation (z.B.: Fehlerausgabe via LED) mit Kunden bzw. Service über einen Kanal (4 Bit). Falls z.B. nach 2maligem Versuch, die Pfandsumme auszugeben, keine positive Bestätigung von der Einheit zurück kommt, gibt Automat auf diesem Kanal ein entsprechendes Muster aus. Bei allen Fehlern wird dabei veranlasst, dass keine neuen Flascheneingaben akzeptiert werden (das Band wird gestoppt) und ebenfalls die Interrupt-Einheit keine Eingaben tätigen kann.

Fehlercode- Komponente liefert Signal, falls es Probleme im Automaten gibt und informiert über den Zustand der aktuelle Flasche : 4 Bit

mögliche Fehler:

- Flasche nicht akzeptabel
- Band klemmt
- Drucker / Kartenleser / externe Einheit nicht i.O.

Ein Kanal steht als Interruptkanal zur Verfügung. Hierüber kann ein spezielles Interrupt-Signal und die folgenden externen Befehle empfangen werden: 4 Bit
Arbeitsweise:

1. Interrupt-Bit-Muster kommt an
2. Befehle (var. Anzahl) kommen und werden verarbeitet solange bis (–8) = '1000' kommt ➔ dann normale Programmarbeit

- „Befehl" ist eine Zahl im 2-er Komplement, die auf die aktuelle Pfandsumme addiert wird
- Wertbereich für diese Daten: -7 bis 7
- Die Eingabe der zu addierenden bzw. subtrahierenden Zahl erfolgt „natürlich" – d.h. als Zahl wie sie der Benutzer eingeben will. Eine dahinterliegende „Magic-Box" zerlegt die eingegebene Zahl in 4-Bit Zahlen, die dann sequentiell geliefert werden. (Negative Zahlen f. Subtraktion im 2-er Komplement)

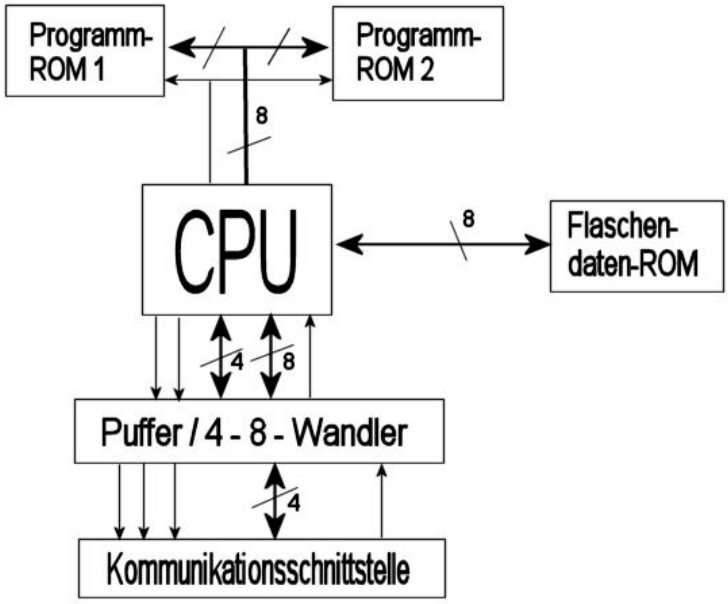
- da die Verarbeitungszeit im Millisekundenbereich liegt, ist bei der Zerlegung von größeren Zahlen kaum eine Verzögerung für den Benutzer erkennbar

(Bandsteuerung bisher weggelassen)

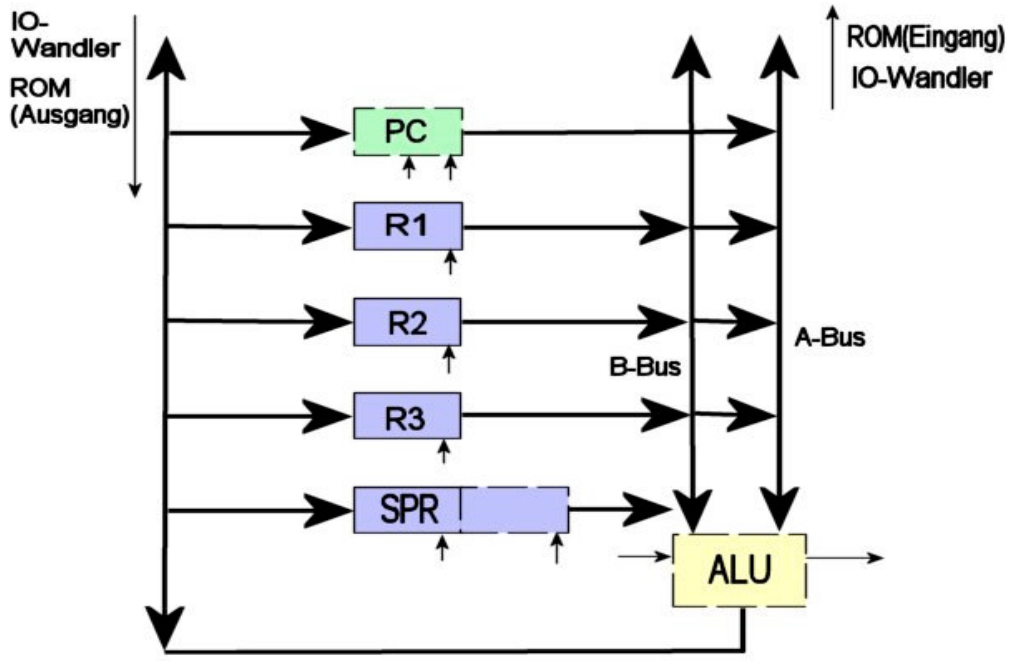
(*) Die Darstellung negativer Zahlen erfolgt im Zweierkomplement. Die Bit-Sequenz ‚1000' wird als „End-Of-Stream" benutzt.

4.Skizzen

8 Bit – Datenleitung zwischen CPU und Puffer ist für den ganz normal Datentransfer.



Ansteuerung der Programm-ROM's: Adresse der nächsten Instruktion wird mit Alu berechnet. D.h. in Register steht jetzt absolute Adresse. Falls bei der Berechnung der neuen Adresse ein Übertrag entstanden ist, wird dieser Übertrag als Flag (CS_ROM) für das anzusprechende Programm-ROM genutzt.



Der links eingezeichnete Bus, muss auch der B-Bus sein ➔ wie soll Steuerwerk sonst damit umgehen.
[mit Sascha und Christoph reden]

A-Bus und B-Bus sind für die Register AUSSCHLIESSLICH Datenausgänge. Der Bus links ist u.a. der Ergebnisbus der ALU und dient als Datenquelle für die Register!