# Computersysteme


**Fragestunde**

Dr.-Ing. Christoph Starke
Institut für Informatik
Christian Albrechts Universität zu Kiel
Tel.: 8805337
E-Mail: chst@informatik.uni-kiel.de

# Kurze Besprechung von Serie 12, Aufgabe 1

## Aufgabe 1

Die alten Ägypter multiplizierten zwei natürliche Zahlen $m$ und $n$ nach folgendem Verfahren:
Zunächst wird das Zwischenergebnis $Erg$ auf 0 gesetzt. Solange $n > 0$ ist, werden anschließend folgende Schritte wiederholt:

- Falls $n$ ungerade ist, bleibt $m$ gleich, $Erg$ wird um $m$ vergrößert und $n$ um eins reduziert.
- Falls $n$ gerade ist, wird $m$ verdoppelt, $n$ halbiert und $Erg$ bleibt gleich.
- Wenn $n = 0$ ist, wird das Verfahren beendet und $Erg$ ist das Ergebnis.

Schreiben Sie ein Assemblerprogramm, das zwei natürliche Zahlen entsprechend dem hier beschriebenen Verfahren multipliziert. Die Faktoren sollen dabei aus dem Speicher ab Adresse 1000 gelesen und das Ergebnis an die nächst größere Speicheradresse geschrieben werden. Sie können dabei davon ausgehen, dass die Faktoren und das Ergebnis kleiner als $2^{30}$ sind.

1. Schritt: Programmablaufplan

# Kurze Besprechung von Serie 12, Aufgabe 1

## 2. Registerbelegung

| Register | Funktion |
|---|---|
| R1,R2 | $m$, $n$<br>In diese Register laden wir unsere Faktoren. |
| R3 | $Erg$<br>In diesem Register steht am Ende das Produkt. |
| R4 | Hilfsregister für Teilbarkeitsprüfung<br>In dieses Register wird das LSB von $n$ kopiert. |

## 3. DLX-Code mit Kommentaren

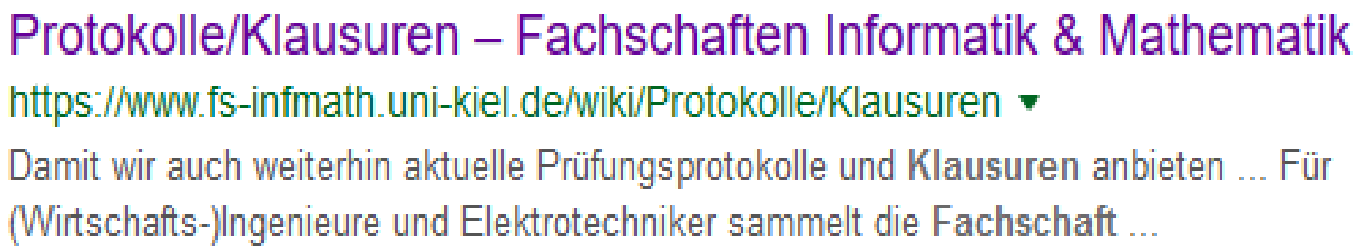
```
1           LW      R1,1000(R0)  / Lade m aus Speicheradresse 1000
2           LW      R2,1004(R0)  / Lade n aus Speicheradresse 1004
3           ADD     R3,R0,R0     / Inistialisiere Erg mit 0.
4  Loop:    BEQZ    R2,End       / Wenn n=0 ist, verzweige zum Programmende.
5           ANDI    R4,R2,#1     / Anderenfalls prüfe LSB=1, also ob n ungerade
6           BNEZ    R4,odd       /  und verzweige ggf. zu odd.
7  even:    SRAI    R2,R2,#1     / Anderenfalls ist n gerade => n wird halbiert
8           SLLI    R1,R1,#1     /  und m verdoppelt.
9           J       Loop         / Wiederhole die Schleife.
10 odd:     SUB     R2,R2,R4     / n ist ungerade und wird dekrementiert
11          ADD     R3,R3,R1     / und Erg wird um m erhöht.
12          J       Loop         / Wiederhole die Schleife.
13 End:     SW      1008(R0),R3  / Ergebnis wird an gespeichert an Adresse 1008.
14          HALT
```

## 4. Programmbeschreibung

… (darauf verzichte ich hier wegen PAP)

**Alte Klausuren**

Google: „klausur fachschaft informatik cau“ =>




Nur aus CAU-Netz aufrufbar.

Dort „Computersysteme“ (z.T. 2 Klausuren unter einem Eintrag)
Weniger relevant: Ältere Klausuren unter „Digitale Systeme“

Weniger relevant als in den älteren Klausuren:
CMOS und MOSFET
Rekursion

Aus dem Inhalt der Fragestunde können Sie keinen Rückschluss auf den Inhalt der Klausuren stellen.

Denn Sie haben die Fragen gestellt und damit den Inhalt der Fragestunde bestimmt.

Nichtsdestotrotz könnten Ihre Fragen größtenteils auch in einer Klausur oder einer mündlichen Prüfung vorkommen.

**Allgemein gilt für alle Aufgaben:**

Auch wenn nicht ausdrücklich nach Rechenwegen, Legenden, Kommentaren, … gefragt ist, müssen Ihre Lösungen eindeutig und leicht nachvollziehbar sein, damit Sie die volle Punktzahl erhalten.

Auch wenn in einer Aufgabe nicht ausdrücklich nach einer Optimierung gefragt ist, sollten Sie eine gute Lösung anstreben. Umständliche Lösungen sind meistens weniger gut, brauchen oft länger in der Bearbeitung und können zu Punktabzug führen.

**Nun zu Ihren konkreten Fragen:**

Wie sollen wir das Horner-Schema aufschreiben?

Wie sollen wir das Horner-Schema aufschreiben?

$$\begin{aligned}(10000000011)_2 &= 1 + 2\,(1 + 2\,(0 + 2\,(0 + 2\,(\ldots(0 + 2)\ldots)))\\ &= 1 + 2 + 2^{10} = (1027)_{10}\end{aligned}$$

# Zahlendarstellung zu $-2^{-142}$ in IEEE (Aufgabe 2a/Serie 3)

# Zahlendarstellung zu $-2^{-142}$ in IEEE (Aufgabe 2a/Serie 3)

**32 Bit (float, single):**

1 Vorzeichenbit

8 Exponentenbits (MSB first)

23 Mantissenbits (MSB first)

Der Wert w einer in diesem Format dargestellten Zahl berechnet sich als:

$w=(-1)^V*(1,M)*2^{E-127}$, falls E>0 und E<255

$w=(-1)^V*(0,M)*2^{-126}$, falls E=0 und M≠0

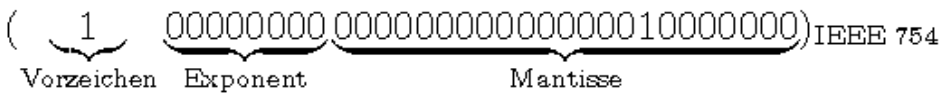
$w=(-1)^V*0$, falls E=0 und M=0

$w=(-1)^v$*Infinity (∞), falls E=255 und M=0

w=NaN (not a number), falls E=255 und M≠0

Wenn alle Exponentenbits im 32-Bit IEEE 754 Format den Wert 0 annehmen, führt dies dazu, dass die "hidden one"verschwindet und die Mantisse mit $2^{-126}$ multipliziert wird. Wird nun das 16. Mantissenbit auf 1 gesetzt, ergibt sich $2^{-142}$.

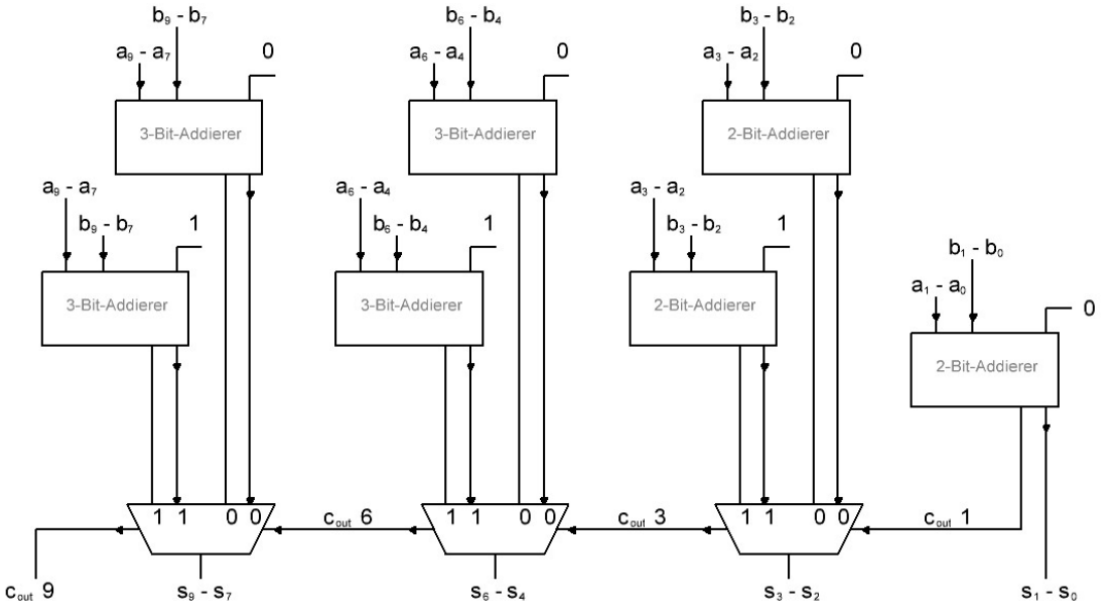
Die 32-Bit IEEE 754 Gleitkommadarstellung ist:



Wie konstruiert man einen Carry-Select-Addierer mit möglichst geringer Schaltzeit (Serie 8)?

# Wie konstruiert man einen Carry-Select-Addierer mit möglichst geringer Schaltzeit (Serie 8)?

Ähnlich wie den 10-Bit-Carry-Select-Addierer aus der Beispiellösung zur Präsenzaufgabe:




Im allgemeinen Fall, beginnend bei LSB: 2-2-3-4-5-6-…

Bestimmen Sie die Kanonische Konjunktive Normalform (KKNF) von

$$f_2 = x_2\overline{x_1} + \overline{x_3}\,x_1\overline{x_0} + \overline{x_3}\,\overline{x_1}\,x_0 + x_3\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

Bestimmen Sie die Kanonische Konjunktive Normalform (KKNF) von

$$f_2 = x_2\overline{x_1} + \overline{x_3}\,x_1\overline{x_0} + \overline{x_3}\,\overline{x_1}\,x_0 + x_3\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

Wertetabelle => Maxterme => Konjunktion

| $x_3$ | $x_2$ | $x_1$ | $x_0$ | $f_2$ | Maxtermfunktionen |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $M_0 = x_3 + x_2 + x_1 + x_0$ |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | 0 | $M_1 = x_3 + x_2 + \overline{x_1} + \overline{x_0}$ |
| 0 | 1 | 0 | 0 | 1 | |
| 0 | 1 | 0 | 1 | 1 | |
| 0 | 1 | 1 | 0 | 1 | |
| 0 | 1 | 1 | 1 | 0 | $M_2 = x_3 + \overline{x_2} + \overline{x_1} + \overline{x_0}$ |
| 1 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | $M_3 = \overline{x_3} + x_2 + x_1 + \overline{x_0}$ |
| 1 | 0 | 1 | 0 | 0 | $M_4 = \overline{x_3} + x_2 + \overline{x_1} + x_0$ |
| 1 | 0 | 1 | 1 | 0 | $M_5 = \overline{x_3} + x_2 + \overline{x_1} + \overline{x_0}$ |
| 1 | 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 0 | $M_6 = \overline{x_3} + \overline{x_2} + \overline{x_1} + x_0$ |
| 1 | 1 | 1 | 1 | 0 | $M_7 = \overline{x_3} + \overline{x_2} + \overline{x_1} + \overline{x_0}$ |

Die kanonische konjunktive Normalform (KKNF) ergibt sich aus der Konjunktion aller Maxterme, für die $f_2$ den Wert logisch 0 annimmt. Somit gilt:

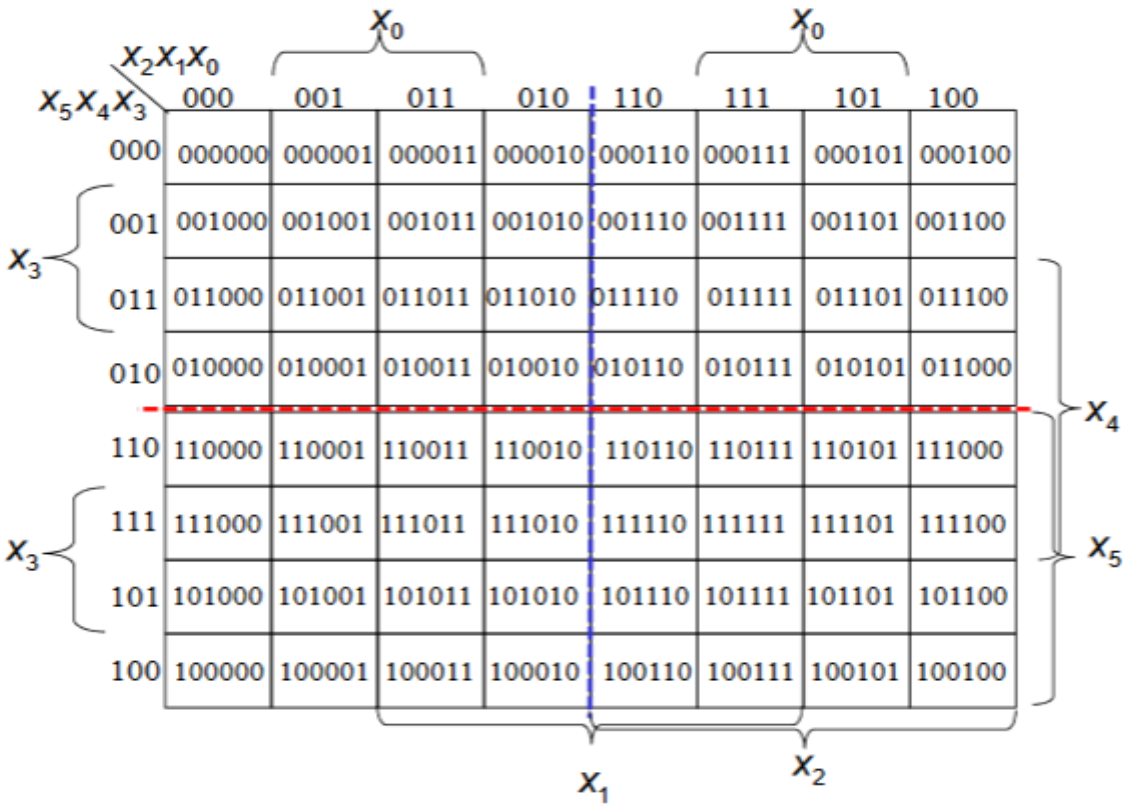
$$f_2 = M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7$$

$$= (x_3 + x_2 + x_1 + x_0) \cdot (x_3 + x_2 + \overline{x_1} + \overline{x_0})$$
$$\cdot (x_3 + \overline{x_2} + \overline{x_1} + \overline{x_0}) \cdot (\overline{x_3} + x_2 + x_1 + \overline{x_0})$$
$$\cdot (\overline{x_3} + x_2 + \overline{x_1} + x_0) \cdot (\overline{x_3} + x_2 + \overline{x_1} + \overline{x_0})$$
$$\cdot (\overline{x_3} + \overline{x_2} + \overline{x_1} + x_0) \cdot (\overline{x_3} + \overline{x_2} + \overline{x_1} + \overline{x_0})$$

Wie wird eine größere KV Tafel mit dem Gray Code aufgestellt und wie werden die Werte der Wertetabelle zugeordnet?

Wie wird eine größere KV Tafel mit dem Gray Code aufgestellt und wie werden die Werte der Wertetabelle zugeordnet?

## KV-Diagramme mit 6 Variablen




6 Variablen haben zwei 3-Bit-Graycodes. Die Variablen $x_0$ und $x_3$ zerfallen in zwei spiegelsymmetrische Komponenten, horizontal und vertikal.

# Wie kommt bei der Bedarfsrolltreppe (Serie 10) auf die Eingänge und den Automatengraphen?

Das Schaltwerk besitzt die folgenden Eingabesignale:

- $LSO$ (Lichtschranke oben), welches genau dann 1 wird, wenn die obere Lichtschranke unterbrochen ist,
- $LSU$ (Lichtschranke unten), welches genau dann 1 wird, wenn die untere Lichtschranke unterbrochen ist,
- $GS$ (Gewichtssensor), welches genau dann 1 wird, wenn der Gewichtssensor eine Person auf der Rolltreppe misst.

Würden diese Eingabesignale so für den Automatengraphen übernommen, würden 3 Bits benötigt. Man stellt fest, dass nur folgende vier Eingaben unterschieden werden müssen, so dass eine Kodierung der Eingabesignale sinnvoll ist:

- Weder eine der beiden Lichtschranken, noch der Gewichtssensor wurden ausgelöst,
- Die obere Lichtschranke wurde ausgelöst, die untere Lichtschranke wurde nicht ausgelöst und der Gewichtssensor wurde ausgelöst oder nicht ausgelöst.
- Die obere Lichtschranke wurde ausgelöst oder nicht ausgelöst, die untere Lichtschranke wurde ausgelöst und der Gewichtssensor wurde ausgelöst oder nicht ausgelöst.
- Keine der beiden Lichtschranken, aber der Gewichtssensor wurde ausgelöst.

## Wie kommt bei der Bedarfsrolltreppe (Serie 10) auf die Eingänge und den Automatengraphen?

Das Schaltwerk besitzt folgende Eingaben, wobei die Angaben in den Klammern Abkürzungen für den Automatengraphen sind und unter den möglichen Eingaben die Codierung mit $X = x_1\ x_0$ steht:

$$X = \{\underbrace{\text{keine Lichtschranken, kein Gewichtssensor}(kLkG)}_{00},$$
$$\underbrace{\text{obere Lichtschranke, nicht untere Lichtschranke}(LO)}_{01},$$
$$\underbrace{\text{untere Lichtschranke}(LU)}_{10},$$
$$\underbrace{\text{keine Lichtschranken, Gewichtssensor}(kLG)}_{11}.$$

Wie kommt bei der Bedarfsrolltreppe (Serie 10) auf die Eingänge und den Automatengraphen?

Mit dem folgenden Schaltnetz erhalten wir x1 und x0:

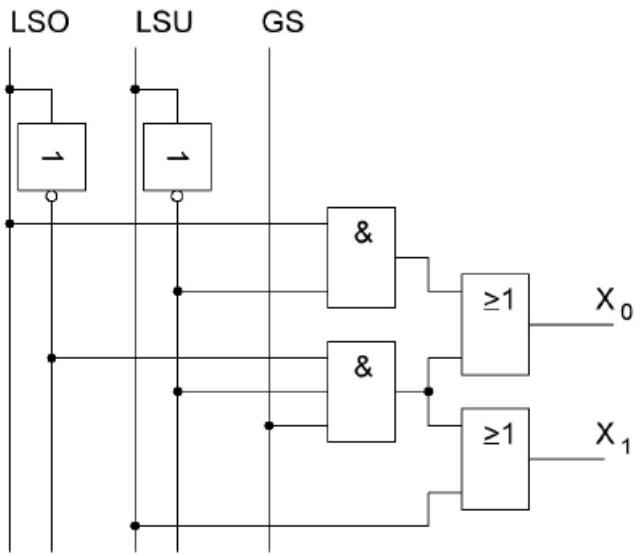



Abbildung 1: Kodierungsschaltnetz der Eingänge

(In der Klausur könnten Sie darauf verzichten, wenn nicht explizit gefordert)

# Wie kommt bei der Bedarfsrolltreppe (Serie 10) auf die Eingänge und den Automatengraphen?

Ausgaben die Codierung mit $Y = R\ H$ steht:

$$Y = \{ \underbrace{\text{Rolltreppe steht still}(nRnH)}_{00},$$
$$\underbrace{\text{Rolltreppe fährt aufwärts}(nRH)}_{01},$$
$$\underbrace{\text{Rolltreppe fährt abwärts}(RnH)}_{10}.$$

Dann entsprechen im folgenden Sinne $R$ und $H$ den Steuerungsignalen für die Rolltreppe:

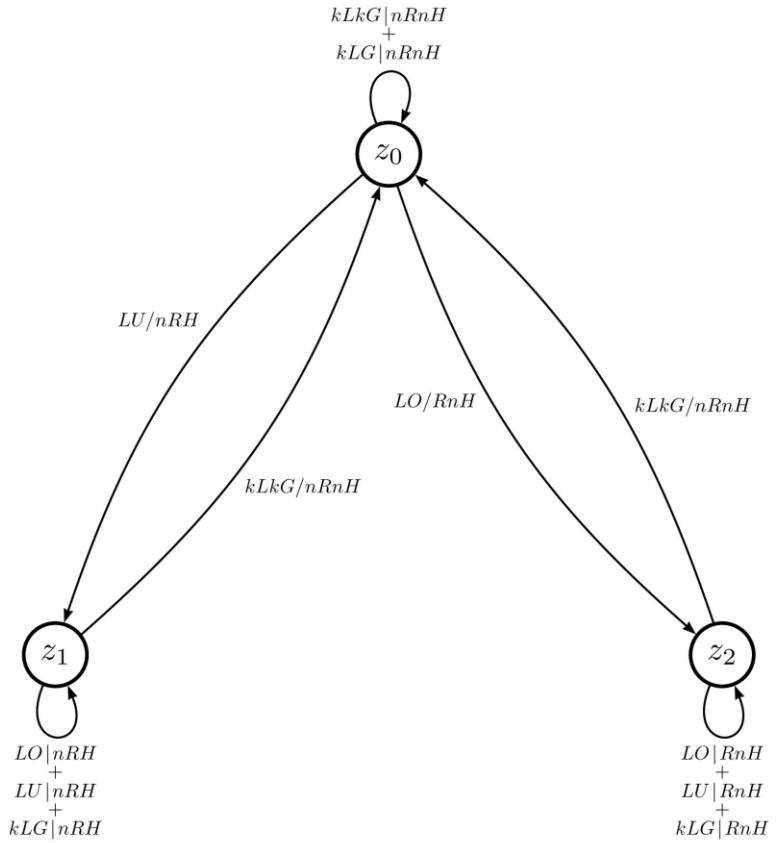
- $R$ (runter), wird genau dann 1 wird, wenn die Rolltreppe abwärts fahren soll,
- $H$ (hoch), wird genau dann 1 wird, wenn die Rolltreppe aufwärts fahren soll.

Das Schaltwerk besitzt folgende Zustände:

- $z_0$: Die Rolltreppe steht still.
- $z_1$: Die Rolltreppe fährt aufwärts.
- $z_2$: Die Rolltreppe fährt abwärts.

Wie kommt bei der Bedarfsrolltreppe (Serie 10) auf die Eingänge und den Automatengraphen?

Daraus ergibt sich folgender Automatengraph:



**Allgemeine Hinweise zu Automatenaufgaben:**

Versuchen Sie, Ein- und Ausgänge sinnvoll zu kodieren, wenn dadurch der Automat mit weniger Ein- bzw. Ausgabebits realisiert werden kann.

Ein Eingabebit weniger reduziert die Wertetabelle um 50%!

Fassen Sie dafür beispielsweise gleichartige Eingabemöglichkeiten zusammen.

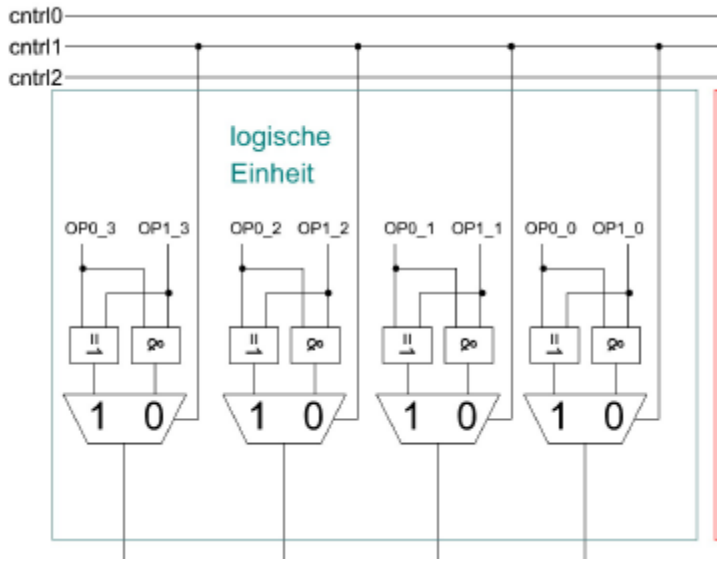
Wenn beispielsweise beim Pizzaautomaten die Extrazutaten Salami, Pepperoni, Oliven, … gleich behandelt werden, müssen diese zusammengefasst werden.

Wenn sich Eingabemöglichkeiten gegenseitig ausschließen, wie z.B. bei der Bedarfsrolltreppe, gibt es oft eine effizientere Codierung.

Entsprechendes gilt für die Ausgänge.

Wie entwirft man eine 4-Bit ALU mit arithmetischer und logischer Einheit (Serie 10)?

Zusätzlich zu der Arithmetik-Einheit aus der Präsenzübung bauen wir noch eine kleine Logik-Einheit:




Durch eine zusätzliche Steuerleitung, die wir aus dem Op-Code abzweigen, wählt ein Datenweg-Mux das Ergebnis der Arithmetik- oder der Logik-Einheit aus.

Bezüglich der Speicheradressierung (Skript S. 138) verstehe ich nicht ganz den Teil über Ausrichtung auf Wortgrenzen. (insbesondere die mittlere Spalte "ausgerichtet auf Byte". Wieso ist ein Byte auf Byte 0,1,2,3,4,5,6,7 ausgerichtet?)

**Ausrichtung auf Wortgrenzen:**

| Adressiertes Objekt | Ausgerichtet auf Byte | Nicht ausgerichtet auf Byte |
|---|---|---|
| Byte | 0,1,2,3,4,5,6,7 | Nie |
| Half word | 0,2,4,6 | 1,3,5,7 |
| Word | 0,4 | 1,2,3,5,6,7 |
| Double word | 0 | 1,2,3,4,5,6,7 |

Mit LB kann man jedes Byte adressieren, mit LH jedes gerade Byte, mit LW jede durch 4 teilbare Adresse und mit LD jede durch 8 teilbare Adresse.

Hinweis: Im Hüser-Tool werden LB, LH, LD nicht unterstützt.

| | | | |
|---|---|---|---|
| LF | F0,50(R3) | Load float | Regs [F0] $\leftarrow_{32}$ Mem [50+Regs [R3] ] |
| LD | F0,50(R2) | Load double | Regs [F0] # #Regs [F1] $\leftarrow_{64}$ Mem[50+Regs [R2] ] |

Was ist bei einem Register-Register Befehl der Unterschied zwischen dem Opcode Teil und dem Function Teil?

Ich hatte es ursprünglich so verstanden, dass alle Befehle (Funktionen) im Opcode kodiert sind, deswegen weiß ich nicht was diese 11 Bit für function genau darstellen/kodieren sollen. (Skript S. 145 bzw. 157)

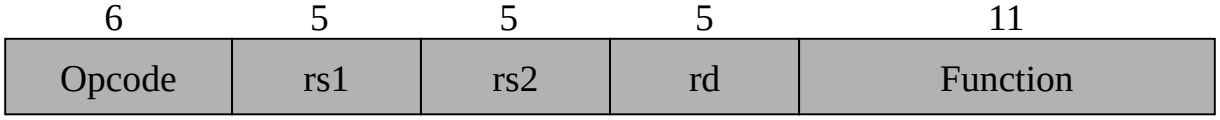
# Befehlsformate

I - Befehl


Typische Immediate-Befehle (rd ← rs1 op immediate)
Loads und Stores von Bytes, Worten, Halbworten (rs1 unbenutzt)

Bedingte Verzweigungen (rs1 : register, rd unbenutzt)
Jump register, Jump and link register
(rd = 0, rs1 = destination, immediate = 0)

R - Befehl




Register-Register ALU Operationen: rd ← rs1 func rs2
func (Function) sagt, was gemacht werden soll: Add, Sub, ...
Read/write auf Spezialregistern und moves

J - Befehl


Jump und Jump and link
Trap und Return from exception

Was ist bei einem Register-Register Befehl der Unterschied zwischen dem Opcode Teil und dem Function Teil? Ich hatte es ursprünglich so verstanden, dass alle Befehle (Funktionen) im Opcode kodiert sind, deswegen weiß ich nicht was diese 11 Bit für function genau darstellen/kodieren sollen. (Skript S. 145 bzw. 157)

Bei den R-Befehlen ist der Opcode 000000.

Die Unterscheidung, ob ADD, SUB, … wird im function-Teil codiert.

Ohne diesen Kniff wären nur 2^6 verschiedene Befehle möglich. Mit diesem Kniff sind 2^6-1 + 2^11 möglich.

Bedeutung von "op" und "func" in den Befehlen der Pipelining-Register in den Pipelining Phasen (letzte VL, Folie 23).

## Pipelining-Register in den Pipeline-Phasen

| Stufe | Was wird alles getan | | |
|---|---|---|---|
| IF | IF/ID.IR ← Mem[PC];<br>IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALU Output} else {PC+4}); | | |
| ID | ID/EX.A ← Regs[IF/ID.$IR_{6..10}$]; ID/EX.B ← Regs[IF/ID.$IR_{11..15}$];<br>ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR;<br>ID/EX.Imm ← $(IR_{16})^{16}$ # # $IR_{16..31}$; | | |
| | **ALU Befehl** | **Load oder store Befehl** | **Verzweigungsbefehl** |
| EX | EX/MEM.IR ← ID/EX.IR;<br>EX/MEM. ALUOutput ←<br>ID/EX. A func ID/EX. B;<br>or<br>EX/MEM.ALUOutput ←<br>ID/EX.A *op* ID/EX.Imm;<br>EX/MEM.cond ← 0; | EX/MEM.IR ← ID/EX.IR<br>EX/MEM.ALUOutput ←<br>ID/EX.A + ID/EX.Imm;<br><br><br><br>EX/MEM.cond ← 0;<br>EX/MEM.B ← ID/EX.B; | <br>EX/MEM.ALUOutput ←<br>ID/EX.NPC+ID.EX.Imm;<br><br><br><br>EX/MEM.cond ←<br>(ID/EX.A *op* 0); |

Im Sinne der vorangegangenen Folien:

Bei den R-Befehlen ist der Befehl im func-Teil codiert, bei I-Befehlen im opcode.

# Was ist ein Mikrocontroller?

# Was ist ein Mikrocontroller?

Ein einfacher Mikrocontroller ist ein Chip mit einem Prozessor und beinhaltet oft auch Arbeits- und Programmspeicher. Also so etwas wie den DLX, den wir nach dem Verstehen der Vorlesung Computersysteme bauen können.

Moderne Mikrocontroller können noch wesentlich mehr Peripherie enthalten, wie z.B. USB- und LAN-Schnittstellen.

Welche Speedup-Formel ist relevant für die Klausur?
In den Folien sind drei gegeben: Speedup (gesamt), Speedup (nach Amdahls Gesetz) und allgemein eine Gleichung namens Speedup.

Antwort: Bei richtiger Anwendung sollten alle 3 Formeln zum selben Ergebnis führen.

Wichtig ist dabei die Unterscheidung zwischen dem Zeitanteil und dem Anteil der Anzahl der Operationen.

Beispiel: 90% R-Befehle, 10% LW-Befehle.

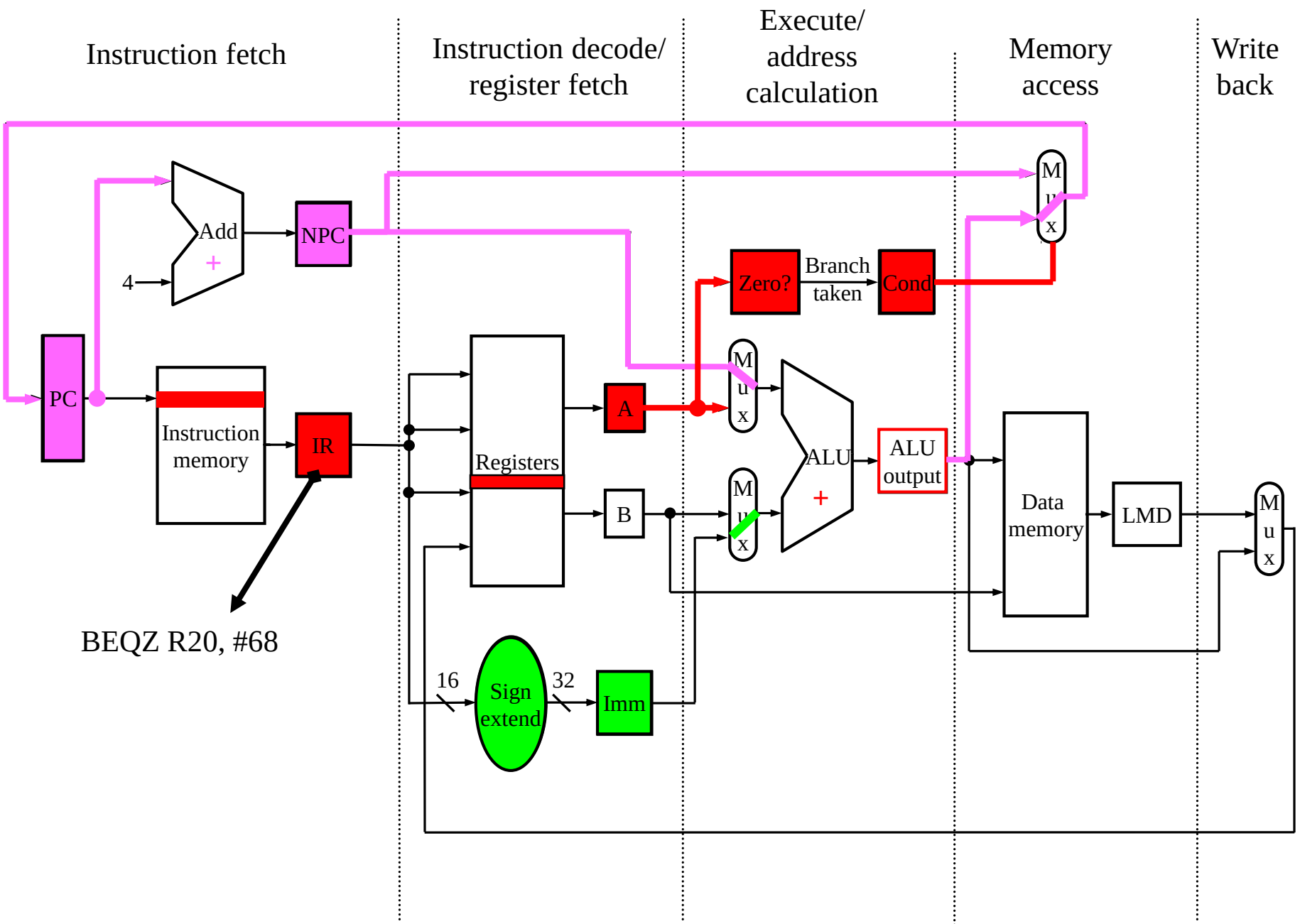
Ein R-Befehl benötigt 2 Takte, ein LW-Befehl 10 Takte.

Der Zeitanteil beträgt

für die R-Befehle 1,8/(1,8+1), also ca. 64%,

für die LW-Befehle 1/(1,8+1), also ca. 36%.

Wie werden Verzweigungsbefehle in den DLX-Taktphasen behandelt, vor allem in der EX & MEM-Phase?




DLX-Datenpfad mit Taktzyklen

Wo werden im DLX-Datenpfad die Labels verarbeitet, wie zum Beispiel ein Label für einen Branch-Befehl?

Wo werden im DLX-Datenpfad die Labels verarbeitet, wie zum Beispiel ein Label für einen Branch-Befehl?

Gar nicht.

Die Labels werden vorher von einem Pre-Compiler in relative Adressen übersetzt:

**J label bedeutet PC <--- PC+4 + offset mit $-2^{25}$ <= offset < $+2^{25}$**

Bitte die alte Klausuraufgabe vorrechnen:

Schreiben Sie ein DLX-Assemblerprogramm, das die Konvertierung einer 32Bit Integer-Zahl im 2er-Komplement in eine 32Bit Gleitkommazahl realisiert.

Die Eingabezahl ist größer als Eins und im Speicher an der Adresse 100 hinterlegt. Die Ausgabe des Programms soll eine entsprechende Gleitkommazahl im IEEE754-Format sein und an der Adresse 104 im Speicher abgelegt werden. Das Runden der Mantisse soll vernachlässigt werden.

…

Die Interpretation einer solchen Zahl lautet: $(-1)^{\text{Vorzeichen}} \cdot (1,\text{Mantisse}) \cdot 2^{\text{Exponent-127}}$

…

Welche Bitfolge steht am Ende Ihres Programms an Adresse 104, wenn zu Beginn des Programms die Dezimalzahl 17 an der Speicheradresse 100 steht?