Hochschule München University of Applied Sciences

Technische Informatik 1

Praktische Übung 5: Mikroprozessor – ein Modellrechner

Prof. Orehek, Prof. Wallentowitz, Prof. Zugenmaier, Prof. Henrici

WiSe 2023/2024

In diesem Praktikum arbeiten 4-5 Zweierteams zusammen, um einen kleinen 4-bit-Modellrechner - verteilt über mehrere Digiboards - aufzubauen. Sie haben zum Aufbau etwa eine Woche Zeit. Beim Praktikumstermin wird der bereits fertig zusammengesteckte und getestete Modellrechner nur noch abgenommen. Für jede Teilgruppe wird es einen Kick-Off Termin geben, an dem die Aufgaben verteilt werden.

Der Modellrechner

Der Befehlssatz des Modellrechners soll folgende Befehle umfassen:

Nr.	Befehl	Opcode	Beschreibung
0	NOP	0000	No Operation
			Lade- und Speicherbefehle
1	LDA #n	0001	Lädt den Akkumulator mit Wert n
2	LDA (n)	0010	Lädt den Akkumulator mit dem Inhalt der Speicherstelle n
3	STA (n)	0011	Speichert den Inhalt des Akkumulators an die Speicherstelle n
			Arithmetik
4	ADD #n	0100	Erhöht den Inhalt des Akkumulators um den Wert n
5	ADD (n)	0101	Erhöht den Inhalt des Akkumulators um den Inhalt an Speicherstelle n
6	SUB #n	0110	Erniedrigt den Inhalt des Akkumulators um den Wert n
7	SUB (n)	0111	Erniedrigt den Inhalt des Akkumulators um den Inhalt an Speicherstelle n
			Sprungbefehle
8	JMP n	1000	Lädt den Programmzähler mit dem Wert n
9	BRZ #n	1001	Addiert n auf den Programmzähler, falls das Zero-Bit gesetzt ist
10	BRC #n	1010	Addiert n auf den Programmzähler, falls das Carry-Bit gesetzt ist
11	BRN #n	1011	Addiert n auf den Programmzähler, falls das Negativ-Bit gesetzt ist

Der schematische Aufbau des Modellrechners ist der Folgende:

Der Prozessor führt jeden Befehl in einem Takt aus. Während *clk* low ist, wird der neue Befehl geholt, bei der Taktflanke von low auf high in das Instruktions- und das Datenregister

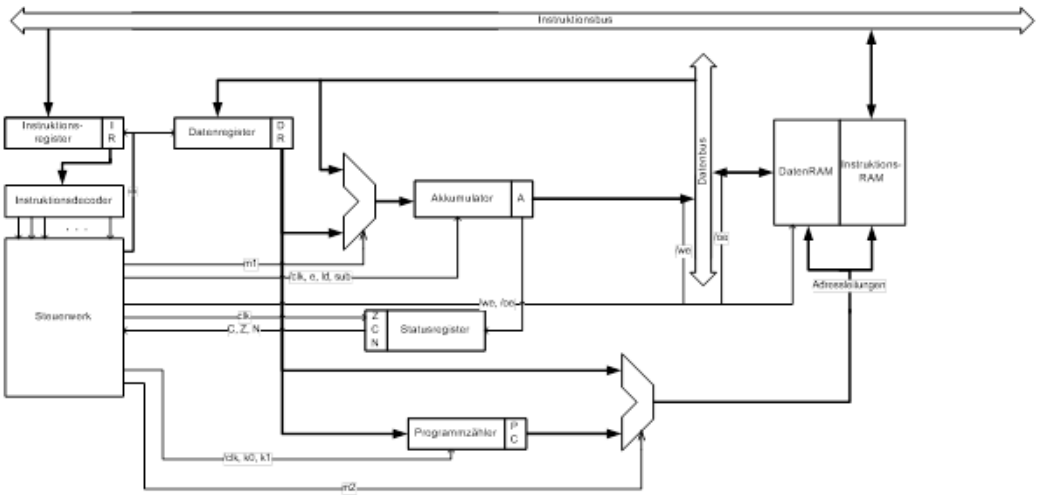


Figure 1: Aufbau des Modellprozessors (nach Hoffmann, Grundlagen der Technischen Informatik)

geschrieben. Während *clk* high ist, wird der Befehl dekodiert, ausgeführt und mit der fallenden Flanke wird geschrieben.

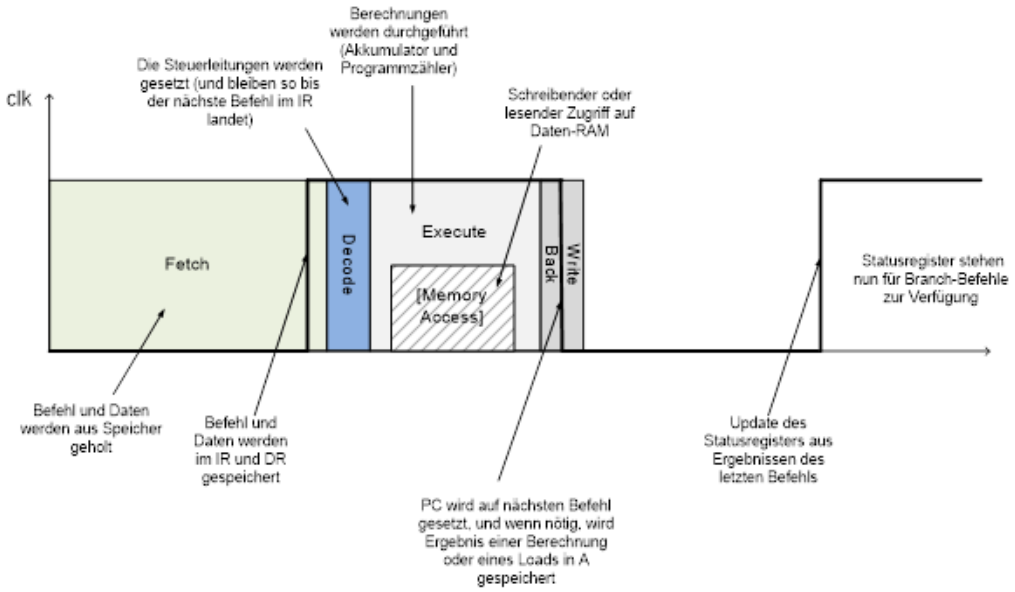


Figure 2: Phasen der Befehlsausführung

Die Phasen *Decode*, *Execute* und *Memory-Access* (sofern notwendig) sind nicht durch Taktflanken definiert, sondern durch die Laufzeiten der Logikgatter.

Nachfolgend werden die unterschiedlichen Phasen beschrieben und erklärt. Dabei wird die Execute-Phase jeweils für die unterschiedlichen Befehle erklärt.

Die Fetch-Phase

Der Programmzähler adressiert Daten- und Instruktions-RAM. Der adressierte Befehl wird in das Instruktionsregister geladen, die Daten in das Datenregister.

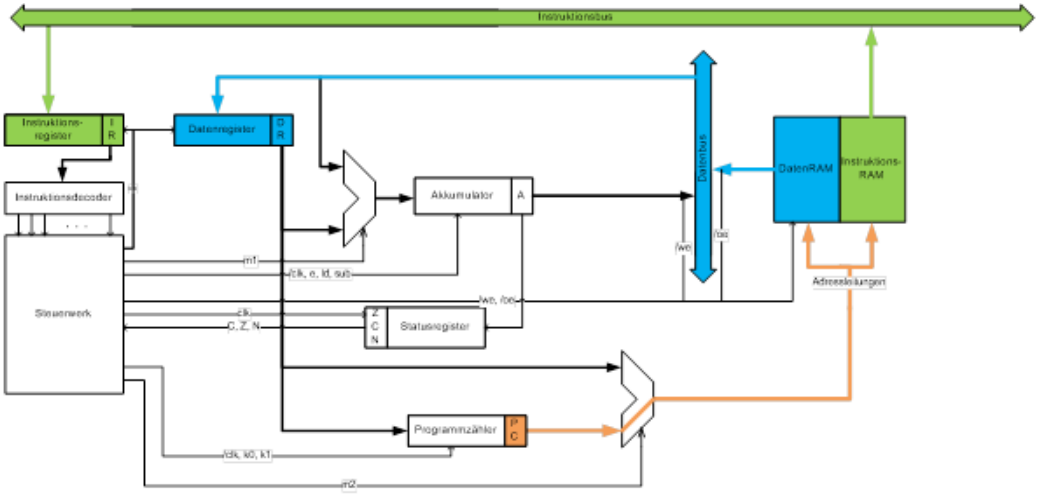


Figure 3: Datenfluss während der Fetch-Phase

Die Decode-Phase

Der im Instruktionsregister geladene Befehl wird im Instruktionsdekoder decodiert. Das Steuerwerk setzt alle Steuerleitungen, um die Ausführung des Befehls zu koordinieren.

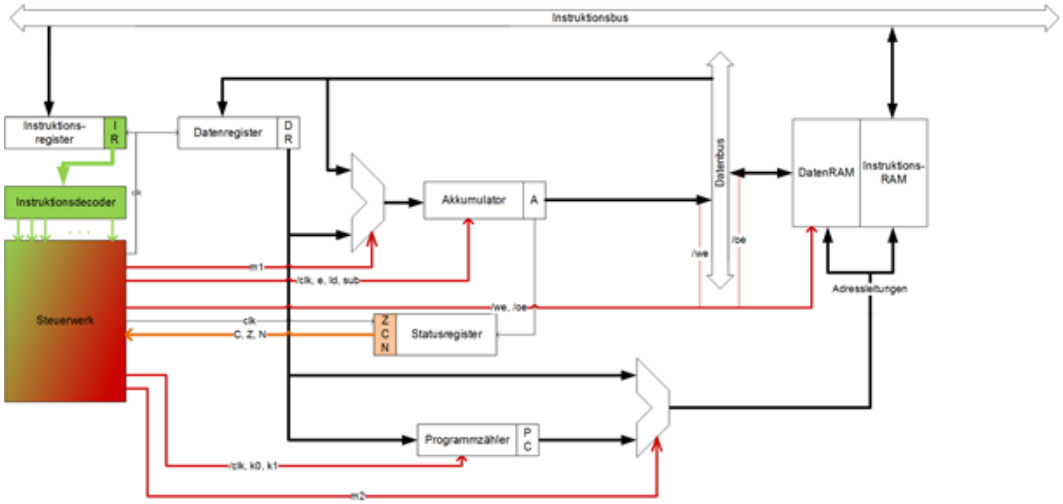


Figure 4: Datenfluss während der Decode-Phase

Execute-Phase (Memory-Access) bei LDA #n, ADD #n, SUB #n

"#n" bezeichnet einen direkt angegebenen Operanden, ein sog. "Immediate". Der Wert im Datenregister geht in den Akkumulator und wird dort verarbeitet.

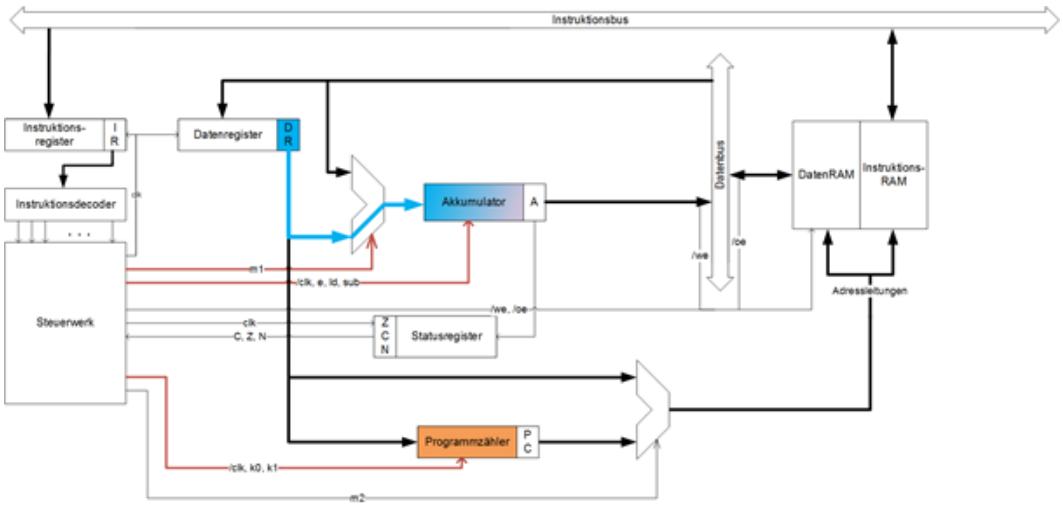


Figure 5: Datenfluss während der Execute-/Memory-Access-Phase bei Ausführen eines Befehls mit direkt angegebenem Operanden

Execute und Memory-Access bei LDA (n), ADD (n), SUB (n)

"(n)" bezeichnet einen Operanden aus dem Speicher. Der zuvor ins Datenregister geladene Wert ist die Adresse, die nun an die Adressleitung des Daten-RAM gelegt wird. Das Daten-RAM gibt den Wert zurück, der an dieser Adresse gespeichert ist. Dieser Wert geht in den Akkumulator und wird dort verarbeitet.

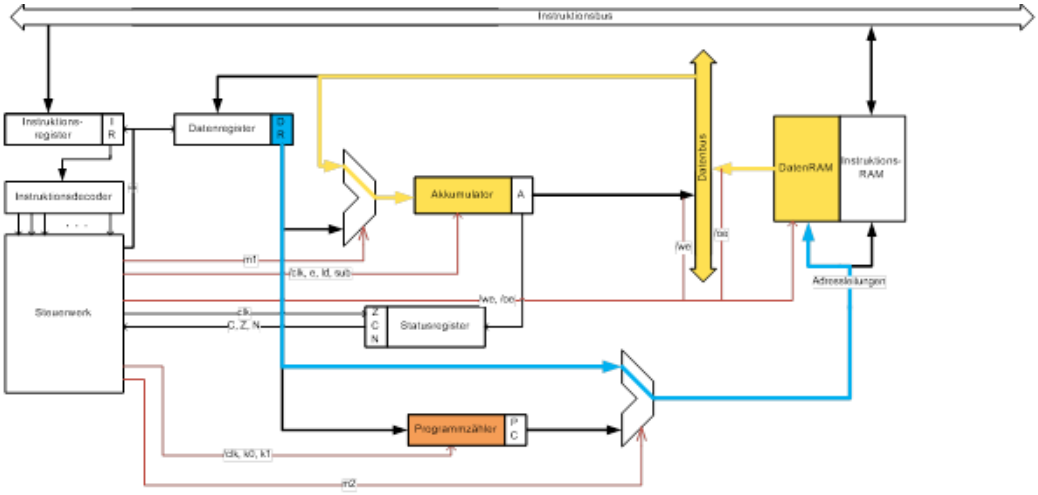


Figure 6: Datenfluss während der Execute-/Memory-Access-Phase bei Ausführen eines Befehls mit Operanden aus dem Speicher

Execute-Phase und Memory-Access bei STA (n)

Der zuvor ins Datenregister geladene Wert ist die Adresse, die nun an die Adressleitung des Daten-RAM gelegt wird. Damit wird die Speicherstelle ausgewählt, an die der Ausgang des Akkumulatorregisters geschrieben wird.

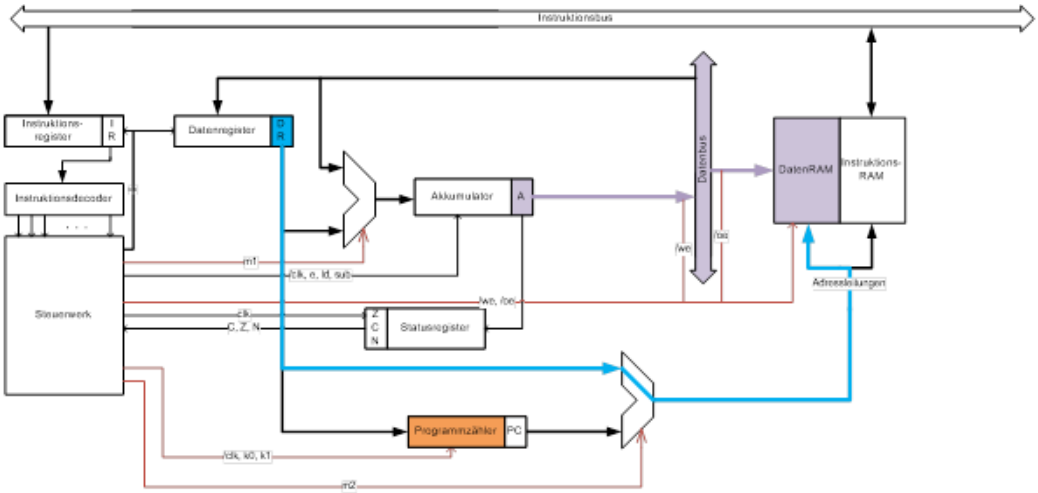


Figure 7: Datenfluss während der Execute-/Memory-Access-Phase beim Befehl Store

Execute-Phase (Memory-Access) bei JMP n, BRZ #n, BRC #n, BRN #n

Der ins Datenregister geladene Wert dient dem Programmzähler als Eingang, mit dem er den nächsten Wert des Programmzählers berechnen kann.

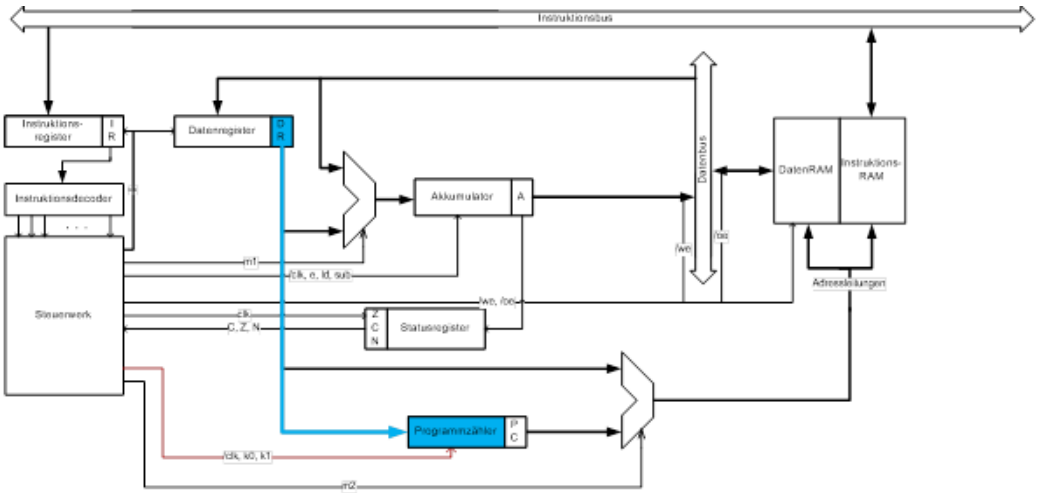


Figure 8: Datenfluss während der Execute-Phase bei Sprungbefehlen

Write-Back bei LDA #n, ADD #n, SUB #n, bzw. LDA (n), ADD (n), SUB (n)

Bei Ausführung eines Befehls, der den Akkumulator verändert, wird bei der fallenden Flanke des Taktes das Ergebnis der internen Verarbeitung im Auffangregister des Akkumulators übernommen und steht dann am Ausgang zur Verfügung. Gleichzeitig wird unabhängig vom aktuellen Befehl bei der fallenden Flanke des Taktes der neue Programm-Counter-Wert im Auffangregister des PC übernommen und steht dann am Ausgang zur Verfügung.

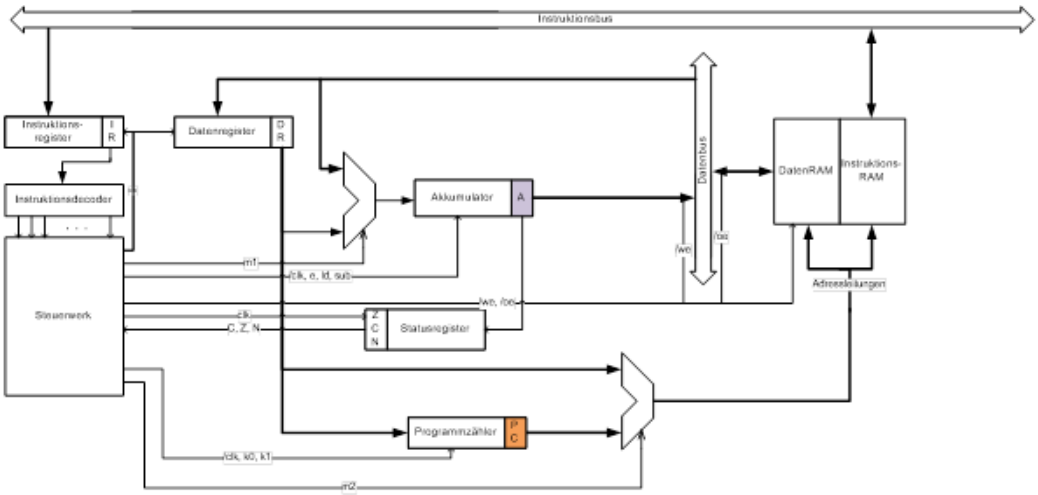


Figure 9: Datenfluss während der Write-Back-Phase

Aufgaben zur Vorbereitung des Praktikums

1. Machen Sie sich das Durchschalten der Transportwege in Abhängigkeit vom ausgeführten Befehl für die Phasen [Fetch] und [Execute + Memory-Access] klar. Tragen Sie in den folgenden Tabellen die benötigten Steuersignale ein [0, 1, -]. Siehe Abbildung 10 für die Funktion der Steuersignale.

Fetch			
Nr.	Befehl	m1	m2
0	NOP		
1	LDA #n		
2	LDA (n)		
3	STA (n)		
4	ADD #n		
5	ADD (n)		
6	SUB #n		
7	SUB (n)		
8	JMP n		
9	BRZ #n		
10	BRC #n		
11	BRN #n		

Execute + Memory Access			
Nr.	Befehl	m1	m2
0	NOP		
1	LDA #n		
2	LDA (n)		
3	STA (n)		
4	ADD #n		
5	ADD (n)		
6	SUB #n		
7	SUB (n)		
8	JMP n		
9	BRZ #n		
10	BRC #n		
11	BRN #n		

2. Schauen Sie sich alle nachfolgenden Teile des Mikroprozessors an. Sie werden für einen der Teile die Verantwortung der Umsetzung übernehmen, müssen am Ende aber alle Teile kennen und verstanden haben.

3. Überlegen Sie sich und dokumentieren Sie geeignete Teststrategien für die einzelnen Teile.

4. Schreiben Sie ein paar Testprogramme für den Modellrechner, mit denen die Befehle getestet werden können. Am Ende müssen alle Befehle getestet werden; für bedingte Sprünge testen Sie bitte sowohl den Fall, dass die Bedingung erfüllt ist, als auch den Fall, dass die Bedingung nicht erfüllt ist. Ein einzelnes Testprogramm soll aus 3-4 Zeilen Code bestehen.

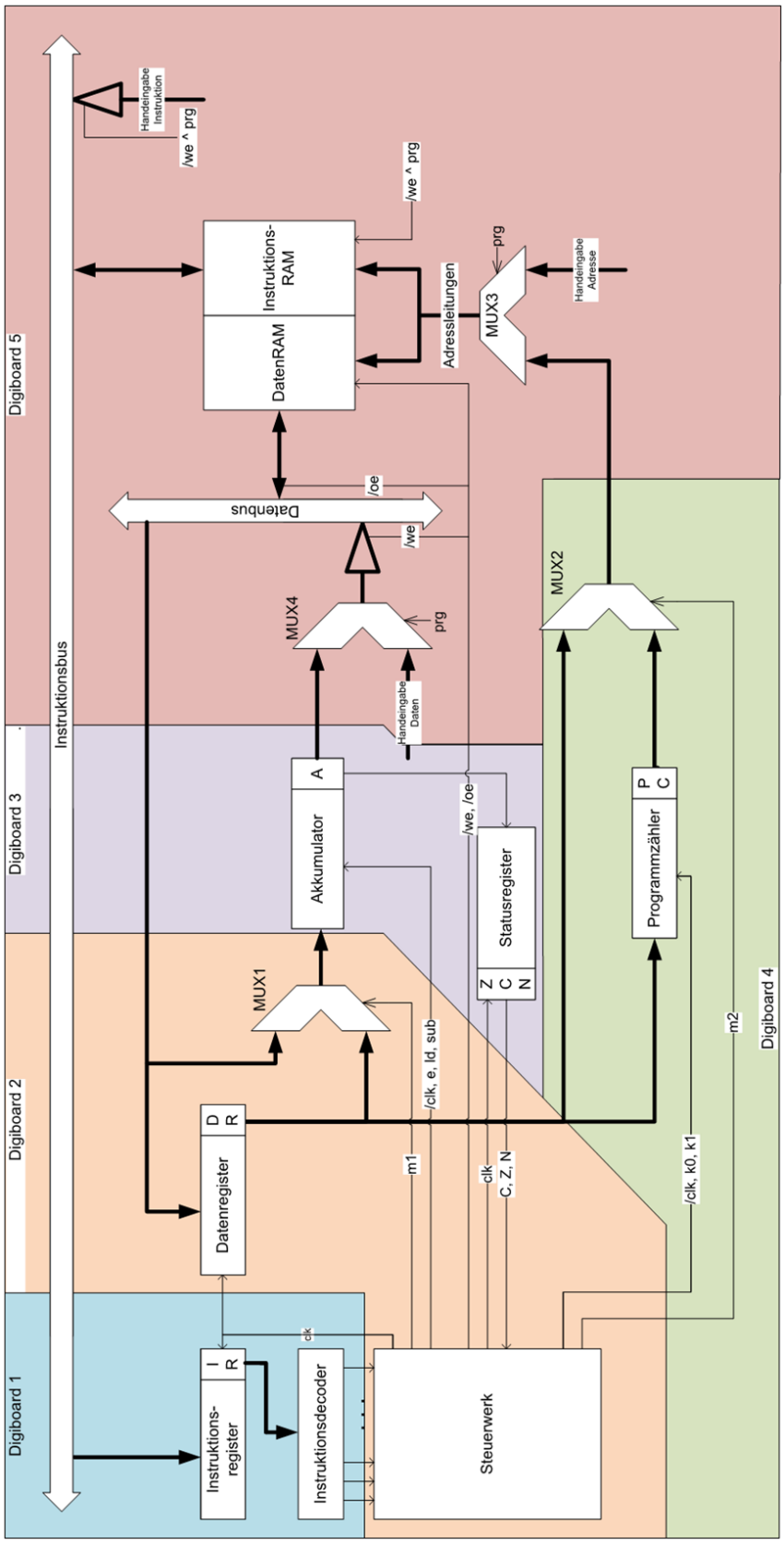


Figure 10: Übersichtsplan mit möglicher Verteilung der Elemente auf die Digiboards

Digiboard 1

Bauen Sie den **Instruktionsdekoder** auf. Nutzen Sie das SRG4 als **Instruktionsregister**.

Der Instruktionsdekoder ist im Wesentlichen ein Demultiplexer. Sie können die Sprungbefehle mit dem Demultiplexer-Baustein (DX) decodieren, für den Rest sind ausreichend Logikgatter vorhanden.

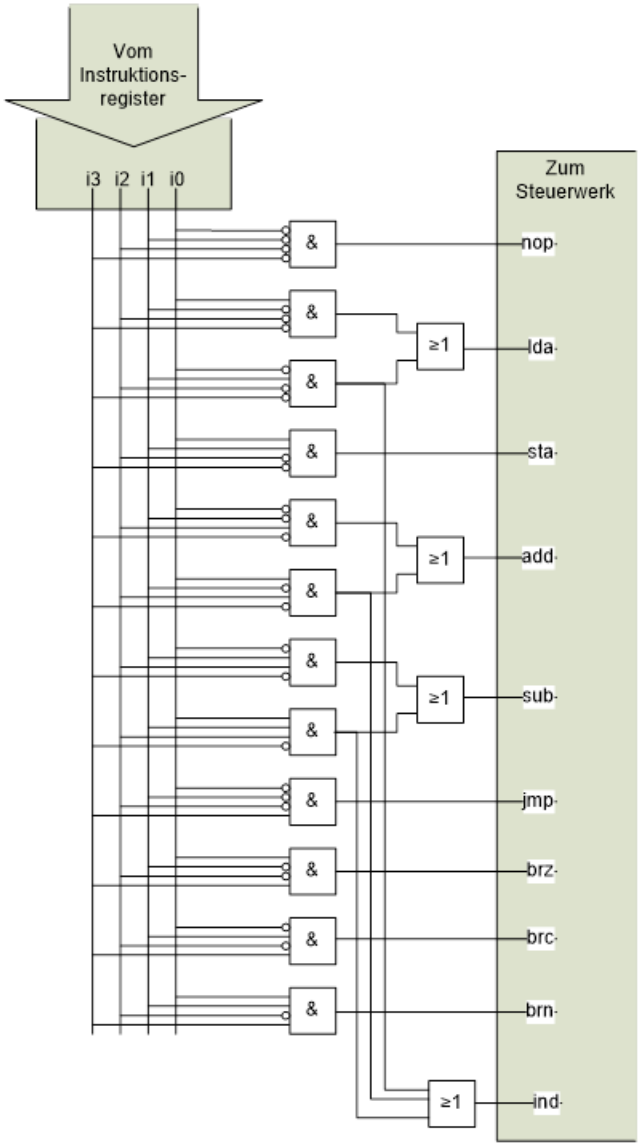


Figure 11: Schematischer Aufbau des Instruktionsdekoders

Digiboard 2

Implementieren Sie einen Taktgeber

Verwenden Sie hierfür den entprellten roten Taster. Ihr Taktgeber soll aus dem einen Taktsignal des Tasters zwei Taktsignale erstellen: eines ist zum **Takten der Register (clk)**, ein weiteres ist zur **Definition der Speicherzugriffsphase beim Schreiben (wclk)**.

Auf den Speicher darf erst schreibend zugegriffen werden, wenn der Ausgang des Steuerwerks und die Werte an den Adressleitungen stabil sind.

Auf den Speicher darf nicht mehr schreibend zugegriffen werden, wenn bereits neue Werte auf Adressleitungen oder Datenbus liegen.

Aus der Zeichnung können Sie ablesen, wie Sie die Signale **clk** und **wclk** erstellen können. Verzögern Sie das Signal des Tasters, am besten indem Sie eine Schaltung bauen, die den Frequenzgenerator und die D-Latches verwendet.

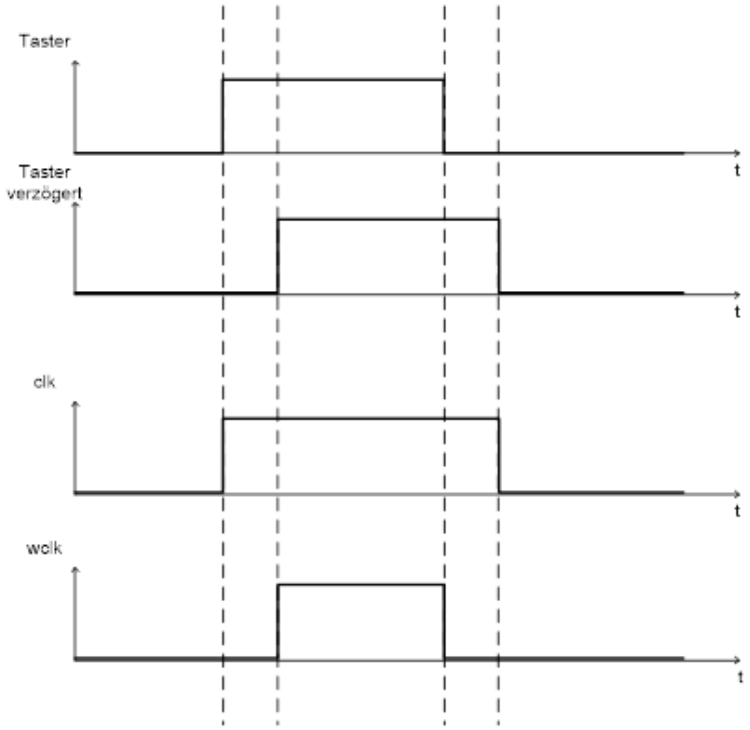


Figure 12: Erzeugung der Taktsignale

Implementieren Sie das Steuerwerk

Das **Steuerwerk** belegt die Steuerleitung für den Akkumulator so, dass dieser die richtige Berechnung mit Eingang und aktuellem Wert durchführt.

Für den *Programmzähler* werden die Steuerleitungen so geschaltet, dass Jumps immer ausgeführt werden; Branches jedoch nur, wenn die Bedingungen erfüllt sind.

Für das RAM wird die Schreibleitung geschaltet. Des Weiteren werden mit den beiden im Übersichtsplan sichtbaren Multiplexern (MUX1, MUX2) die Transportwege korrekt durchgeschaltet.

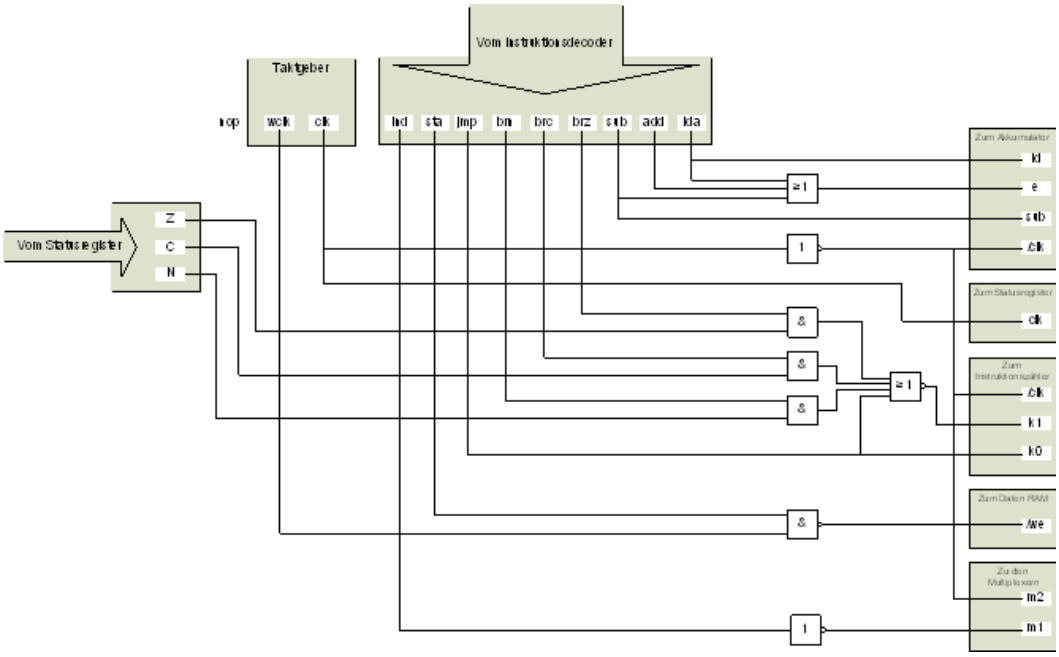


Figure 13: Schematischer Aufbau des Steuerwerks

Implementieren Sie das Datenregister und MUX1

Auf diesem Digiboard soll auch das **Datenregister** mit dem angebundenen **Multiplexer** von RAM bzw. Datenregister in Richtung Akkumulator implementiert werden (MUX1).

Hinweise: Sie können die ALU auch als MUX einsetzten. Die Beschreibung der ALU ist am Ende dieser Angabe zu finden. Diese ist korrekter als die in der Beschreibung des Digiboards angegebene.

Digiboard 3

Implementieren Sie den Akkumulator

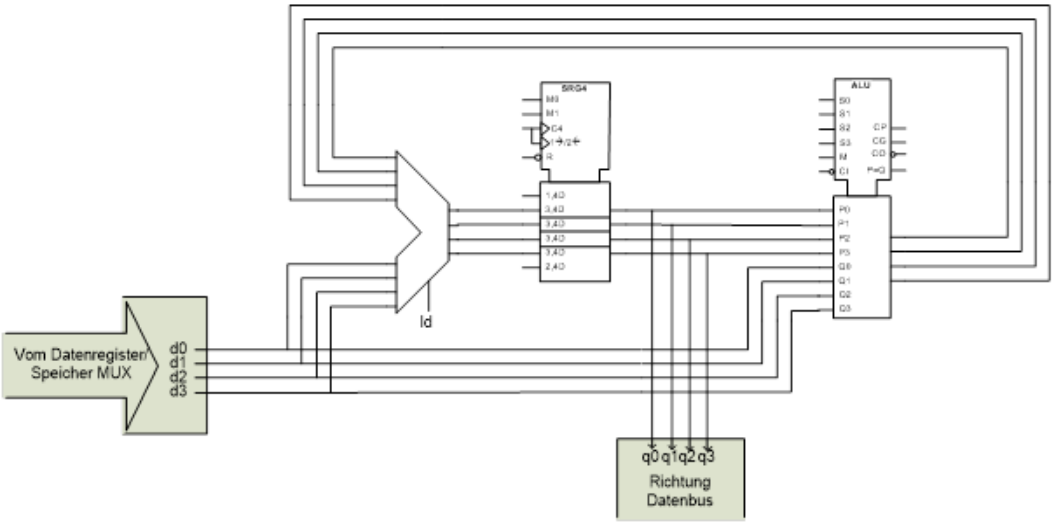


Figure 14: Schematischer Aufbau des Akkumulators

Machen Sie sich den Datenfluss im **Akkumulator** für die verschiedenen Befehle klar.

Vereinfachen Sie den Akkumulator unseres Modellprozessors zuerst so weit wie möglich, indem Sie die in der ALU eingebauten Funktionen verwenden.

Entwickeln Sie die notwendige Steuerlogik mit einem Minimum an logischen Gattern.

Die folgende Tabelle beschreibt die in den Akkumulator hineingehenden Steuerleitungen:

/clk		die Register des Akkumulators schalten auf der fallenden Flanke der Clock (Write-Back-Phase)
ld	ld=0	der Akkumulator führt eine Berechnung durch
	ld=1	das Akkumulatorregister wird mit den an d0-d3 anliegenden Werten geladen
sub	sub=0	der Akkumulator addiert d0-d3 auf den aktuellen Registerinhalt
	sub=1	der Akkumulator subtrahiert d0-d3 vom aktuellen Registerinhalt
e	e=0	das Akkumulatorregister ist in der Write-Back-Phase inaktiv (wird nicht überschrieben)
	e=1	das Akkumulatorregister ist in der Write-Back-Phase aktiv (wird überschrieben)

Erweitern Sie das Akkumulatorregister so, dass auch das Carry-Bit aus der letzten Operation gespeichert wird. Hierzu können Sie eines der RSJK-Flipflops verwenden. Auch dieses Carry-Bit benötigt ein Enable.

Implementieren Sie das **Statusregister**, wobei Sie anstelle der D-Flipflops die JK-Flipflops verwenden (das Carry-Bit wird sowohl im Akkumulator als auch im Status-Register gespeichert):

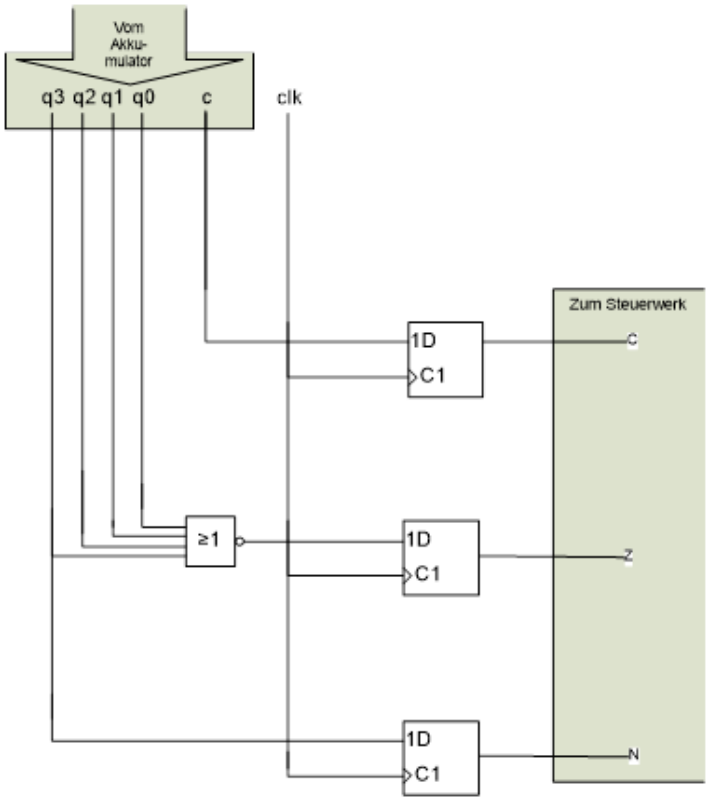


Figure 15: Schematischer Aufbau der Statusregister

Achtung: Statusregister und Akkumulatorregister schalten auf unterschiedlichen Flanken.

Digiboard 4

Implementieren Sie den Programmzähler und den Multiplexer (MUX2)

MUX2 definiert die Speicheradresse entweder durch das Datenregister oder den Programmzähler.

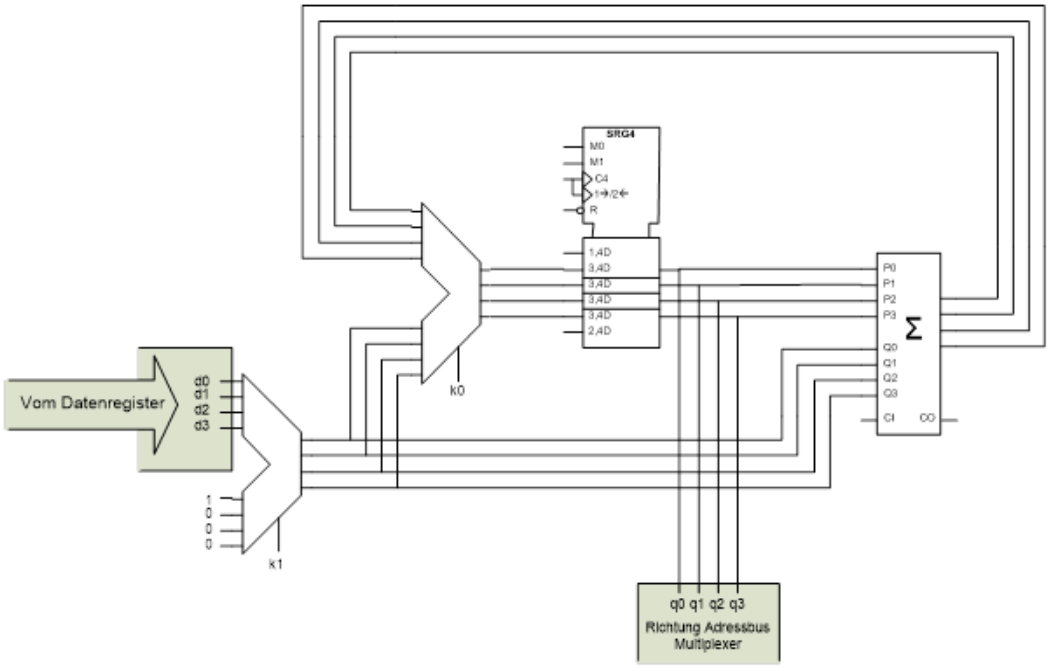


Figure 16: Schematischer Aufbau des Programmzählers

Machen Sie sich den Datenfluss im Programmzähler für die verschiedenen Befehle klar.

Vereinfachen Sie den Programmzähler unseres Modellprozessors zuerst so weit wie möglich, indem Sie anstelle der beiden Multiplexer die in der ALU eingebauten Funktionen verwenden.

Entwickeln Sie die notwendige Steuerlogik mit einem Minimum an logischen Gattern. Der Programmzähler soll in Abhängigkeit der in diesen hineingehenden Steuerleitungen **k0** und **k1** folgende Funktionen unterstützen:

k0	**k1**	**Funktion**
0	0	relativer Sprung
0	1	nächster Befehl
1	0	absoluter Sprung
1	1	kommt nicht vor

Digiboard 5

Verwenden Sie den RAM-Speicherbaustein für das Daten-RAM, das "EEPROM" für den Instruktionsspeicher.

Implementierung der Programmierlogik

Implementieren Sie die **Programmierlogik**. Diese dient dazu, per Hand den Speicher zu beschreiben. Der folgende Übersichtsplan wurde um die Programmierlogik erweitert.

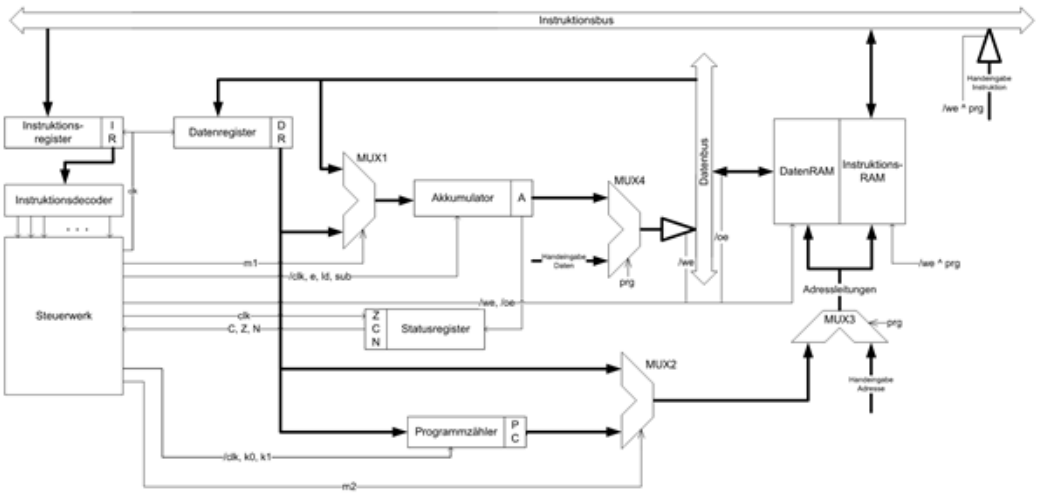


Figure 17: Übersichtsplan mit Programmierlogik

Einen Schalter eines anderen Boards können Sie verwenden, um in den Programmiermodus zu wechseln.

Es soll möglich sein, mit Hilfe des Codierschalters die zu schreibende Adresse einzustellen. Ein 4-bit 2:1-Multiplexer soll verwendet werden, um zu wählen, ob die Adresse vom Adressbus oder vom Codierschalter kommen soll (MUX3).

Verwenden Sie einen 4-bit 2:1-Multiplexer, um zwischen zu schreibenden Daten vom Akkumulator oder von den Schaltern der Programmierlogik zu wechseln (MUX4).

Nutzen Sie zwei 3-State-Bus-Treiber (z.B. ein 74244 auf separat auf das Digiboard aufgestecktem Nullkraftsockel), um die Programmierlogik und den Akkumulator (d.h. Ausgang des Multiplexers für die Daten, Ausgang der Schalter für die Instruktionen) außerhalb des Schreibvorgangs vom Daten- und Instruktionsbus zu entkoppeln. Der Nullkraftsockel hat 28 Pins, während der 3-State-Treiberbaustein (IC) nur 20 hat. Daher stimmt die Nummerierung auf dem Sockel nicht immer mit der des ICs überein. Die Nummerierung stimmt überein für 1-10, hingegen sind die Pins 11-20 am IC an den Ausgängen des Sockels mit Nummern 19-28 versehen.

Achtung: Wenn Sie die Stromversorgung am Baustein nicht korrekt beschalten, geht dieser kaputt. Fragen Sie lieber einmal zu viel nach!

Verwenden Sie den Taster, um den eingestellten Wert in den Speicher zu schreiben.

Stellen Sie sicher, dass bei laufendem Mikroprozessor (also nicht im Programmiermodus) bei *clk* low vom Mikroprozessor keine Daten in den Speicher geschrieben werden.

Hier finden Sie eine Beispielimplementierung der Programmierlogik:

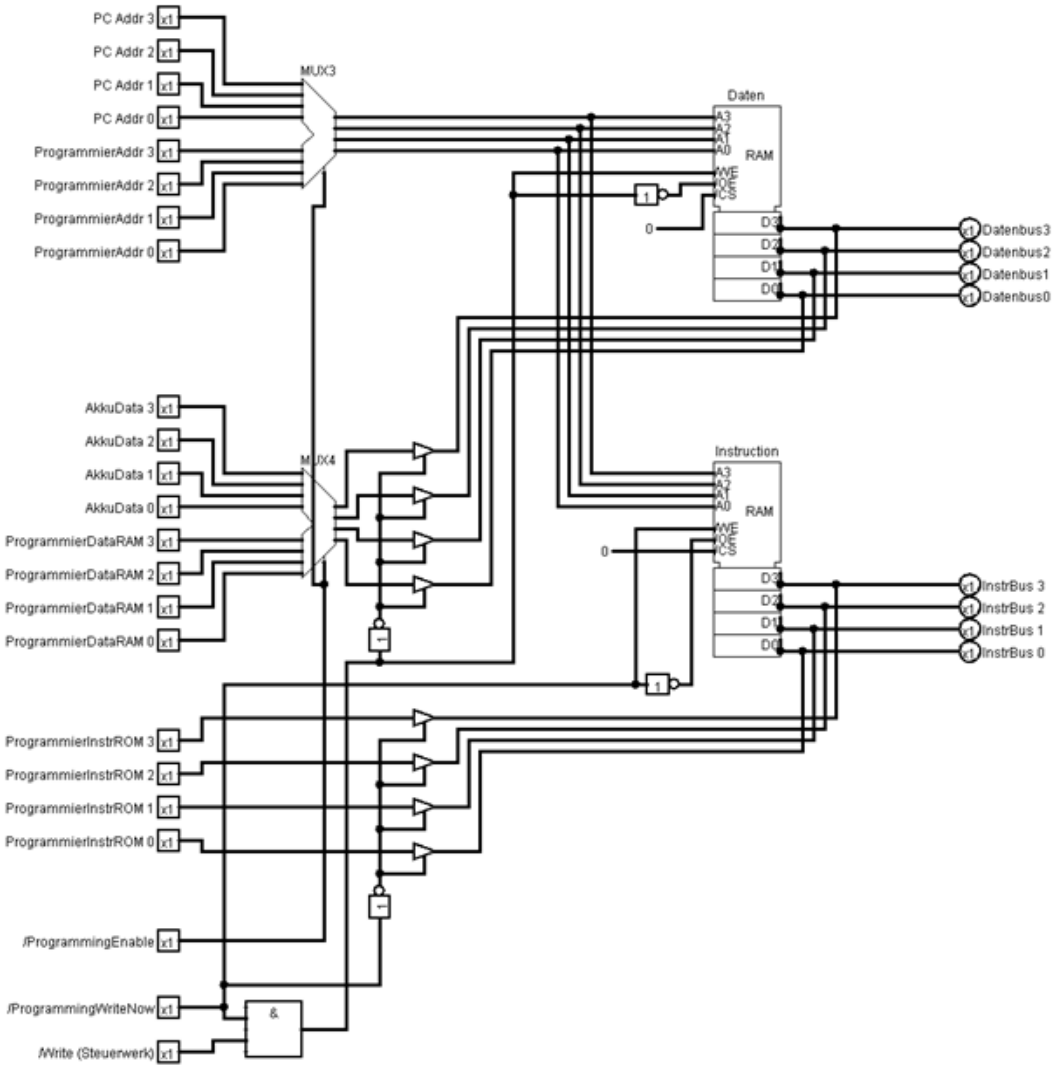


Figure 18: Beispiel für Programmierlogik

Testen

Jede Gruppe testet Ihre eigene Schaltung ausgiebig, bevor diese zum Mikroprozessor zusammengeschlossen wird! Dokumentieren Sie, was Sie getestet haben und wie. Zwei weitere Gruppen testen ebenfalls das Board und zeichnen ab, dass sie dies getan haben.

Schließen Sie den Mikrocontroller zusammen und testen Sie anschließend jeden Befehl mit einem kleinen Programm. Überlegen Sie, was Sie erwarten und testen erst dann. Kopfloses Umstecken von Leitungen hilft nicht bei der Fehlerbehebung!

Dokumentieren Sie jedes Programm und dessen Ablauf als Tabelle. Eine Vorlage dafür finden Sie weiter hinten in dieser Dokumentation. Die Tabelle beinhaltet die Inhalte der Register und der Speicherstellen, die im Laufe des Programms beschrieben werden. Verwenden Sie für jedes Testprogramm eine eigene Tabelle – diese können Sie beliebig oft ausdrucken.

Denken Sie daran, bei bedingten Sprüngen beide Varianten zu testen.

Achten Sie darauf, an welchen Flanken welche Register schalten sollen und an welchen Sie tatsächlich schalten.

Tipps zur Fehlerbehebung

- Verbinden Sie die Massen aller Boards sternförmig (keine Schleifen der Masseleitungen).
- Manchmal ist die Signalqualität des Takts so schlecht, dass das Register doppelt schaltet. Dies fällt insbesondere am PC und am Akkumulator auf. Wenn dies passiert, können Sie das Taktsignal auf dem entsprechenden Digiboard erst durch den Monoflop leiten oder durch ein freies Gatter. Bedenken Sie, dass Sie dies auf allen Boards gleich machen, damit die daraus resultierende Verzögerung des Signals auf allen Boards gleich ist.
- Wenn die Register auf beiden Flanken schalten, dann probieren Sie die Taktverteilung mit kürzeren Kabeln oder mit anderer Kabelführung. Manchmal hilft auch, das Taktsignal mit 1-3 LEDs zu belasten (dann schwingt das Signal nicht mehr über den Schwellwert zum Auslösen des Registers). Dokumentieren Sie aber in einem solchen Fall präzise, was passiert ist und was Sie dagegen unternommen haben.

Für die Abnahme

Bevor Sie zur Abnahme "rufen", stellen Sie sicher, dass Sie Ihre **gesamte Dokumentation geordnet und griffbereit** haben.

Bereiten Sie ein Programm für die Abarbeitung vor.

Programm

Adresse	Befehl	Instruktion	Daten
0			
1			
2			
3			
4			
5			
6			
7			

Schritt	*clk*	PC	Adressbus	Datenbus	IR	DR	A	SR C Z N	Adresse	Inhalt
1	0									
	1									
2	0									
	1									
3	0									
	1									
4	0									
	1									
5	0									
	1									
6	0									
	1									
7	0									
	1									
8	0									
	1									
9	0									
	1									
10	0									
	1									
11	0									
	1									
12	0									
	1									
13	0									
	1									
14	0									
	1									

"Adresse" und "Inhalt" bei Befehlen, die aus dem Speicher laden bzw. in Speicher schreiben (LDA (n), ADD (n), STA (n)).

Programm

Adresse	Befehl	Instruktion	Daten
0			
1			
2			
3			
4			
5			
6			
7			

Schritt	*clk*	PC	Adressbus	Datenbus	IR	DR	A	SR C Z N	Adresse	Inhalt
1	0									
	1									
2	0									
	1									
3	0									
	1									
4	0									
	1									
5	0									
	1									
6	0									
	1									
7	0									
	1									
8	0									
	1									
9	0									
	1									
10	0									
	1									
11	0									
	1									
12	0									
	1									
13	0									
	1									
14	0									
	1									

"Adresse" und "Inhalt" bei Befehlen, die aus dem Speicher laden bzw. in Speicher schreiben (LDA (n), ADD (n), STA (n)).

Programm

Adresse	Befehl	Instruktion	Daten
0			
1			
2			
3			
4			
5			
6			
7			

Schritt	*clk*	PC	Adressbus	Datenbus	IR	DR	A	SR C Z N	Adresse	Inhalt
1	0									
	1									
2	0									
	1									
3	0									
	1									
4	0									
	1									
5	0									
	1									
6	0									
	1									
7	0									
	1									
8	0									
	1									
9	0									
	1									
10	0									
	1									
11	0									
	1									
12	0									
	1									
13	0									
	1									
14	0									
	1									

"Adresse" und "Inhalt" bei Befehlen, die aus dem Speicher laden bzw. in Speicher schreiben (LDA (n), ADD (n), STA (n)).

Programm

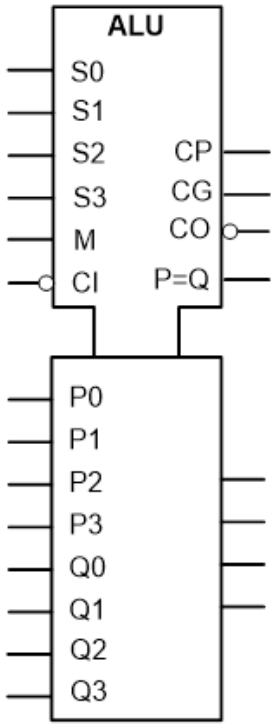
Adresse	Befehl	Instruktion	Daten
0			
1			
2			
3			
4			
5			
6			
7			

Schritt	*clk*	PC	Adressbus	Datenbus	IR	DR	A	SR C Z N	Adresse	Inhalt
1	0									
	1									
2	0									
	1									
3	0									
	1									
4	0									
	1									
5	0									
	1									
6	0									
	1									
7	0									
	1									
8	0									
	1									
9	0									
	1									
10	0									
	1									
11	0									
	1									
12	0									
	1									
13	0									
	1									
14	0									
	1									

"Adresse" und "Inhalt" bei Befehlen, die aus dem Speicher laden bzw. in Speicher schreiben (LDA (n), ADD (n), STA (n)).

Beschreibungen von Bausteinen

Rechenwerkbaustein (ALU)



Anschlussbelegung

- **P0 ... P3:** 4-Bit-Eingangsdaten (P-Operand)
- **Q0 ... Q3:** 4-Bit-Eingangsdaten (Q-Operand)
- **S0 ... S3:** Funktionsauswahl über das 4-Bit-Steuerwort an den Eingängen. Mit S0 ... S3 wird die Operation festgelegt.
- **M:** Auswahlleitung für die Operationsart (Mode-Control), Umschaltung arithmetische / logische Operation
- **CI:** Übertragseingang (Carry in)
- **CO:** Übertragsausgang (Carry out)
- **CP, CG:** Carry Propagate, Carry Generate
- **P=Q:** Komparatorausgang. Dieser Ausgang führt 1-Signal, wenn alle Ergebnisausgänge 1-Signal führen. Diese Funktion kann bei bestimmten Operationen benutzt werden, um Gleichheit der Operanden A und B festzustellen.

Die folgende Tabelle enthält die wichtigsten Operationen und Steuersignale für den ALU-Baustein.
Hinweise zur Anschlussbelegung

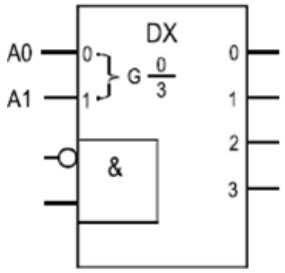
- CI hat keinen Einfluss auf logische Operationen.
- Operand P entspricht der Information an den Eingängen P.
- Operand Q entspricht der Information an den Eingängen Q.

$S0$	$S1$	$S2$	$S3$	M	$\overline{CI}$	Ergebnis
0	0	0	0	0	0	$P+1$
0	1	1	0	0	0	$P-Q$
0	1	1	0	0	1	$P-Q-1$
1	0	0	1	0	1	$P+Q$
1	0	0	1	0	0	$P+Q+1$
0	0	1	1	0	1	$P+P$
0	0	1	1	0	0	$P+P+1$
1	1	1	1	0	1	$P-1$
0	0	0	0	1	X	$\overline{P}$
1	0	1	0	1	X	$\overline{Q}$
0	1	1	0	1	X	$P \oplus Q$ (d.h. xor)
1	0	0	1	1	X	$\overline{P \oplus Q}$
0	1	0	1	1	X	Q
1	1	0	1	1	X	$P \wedge Q$

$S0$	$S1$	$S2$	$S3$	M	$\overline{CI}$	Ergebnis
0	1	1	1	1	X	$P \vee Q$
1	1	1	1	1	X	P

Demultiplexer

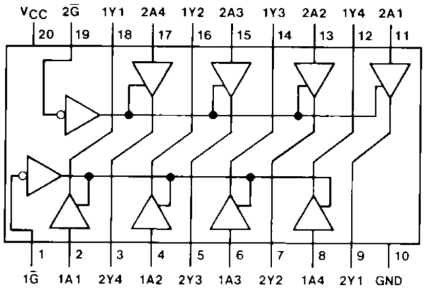
4-Kanal-Demultiplexer mit zwei Eingängen, davon einer invertiert. Die Tabelle zeigt die Belegung der Steuereingänge **A0** und **A1** für den jeweiligen Ausgangskanal.



A0	A1	Kanal Channel
0	0	0
0	1	1
1	0	2
1	1	3

3-State Bus Driver / Receiver 74LS244

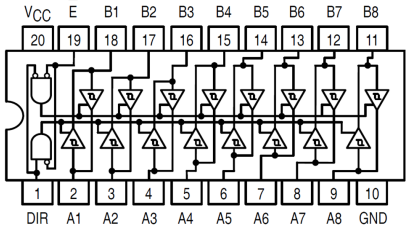
Zwei 4-bit 3-State-Bustreiber in einem Gehäuse. Es werden jeweils die Eingänge von einer Seite auf die andere durchgeschaltet. Wenn das Gate ($1\overline{G}$ bzw. $2\overline{G}$) auf low geschaltet wird, ist der Bustreiber durchlässig, bei Beschaltung mit high geht der Ausgang in den Hochimpedanz-Zustand. **Achtung: Wenn Sie die Stromversorgung falsch anschließen, geht der Baustein kaputt.**
Fragen Sie lieber einmal zuviel nach, oder lassen Sie die Beschaltung des Bausteins mit der Stromversorgung von einem Tutor verifizieren, bevor Sie das Digiboard anschalten.



$\overline{G}$	A	Output Y
L	L	L
L	H	H
H	X	Z

Octal Bus Transceiver 74LS245

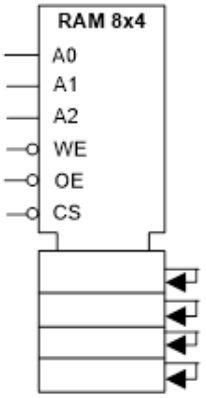
Ein 8-bit-Tri-State-Bustreiber, der sowohl von Bus A nach Bus B, als auch von B nach A betrieben werden kann, festgelegt durch Eingang DIR. $\overline{E}$ auf high schaltet die Ausgänge auf Tri-State.



$\overline{E}$	DIR	Output
L	L	Bus Data A to Bus B
L	H	Bus Data B to Bus A
H	X	Isolation (high impedance)

RAM / EEPROM

Tatsächlich handelt es sich bei beiden Bausteinen um ein statisches RAM, wobei der EEPROM genannte Baustein mit einem Kondensator gepuffert ist und bei Ausschalten des Boards einige Minuten lang den Inhalt behält.
Das Chip-Select Signal ist low-aktiv, d.h. wenn $\overline{CS}$ auf low gelegt wird, wird dieser Chip aktiviert.
Ist $\overline{CS}$ high, werden alle Eingangssignale ignoriert und die Ausgänge gehen auf hohe Impedanz.
Das Write-Enable Signal ist low-aktiv, d.h. der Write-Enable-Eingang $\overline{WE}$ muss zum Lesen auf high und zum Schreiben auf low sein. Solange Write-Enable auf low ist, werden die anliegenden Daten an die Adresse $(A_3)A_2A_1A_0$ geschrieben. Wenn Write-Enable auf high ist (Lesemodus), können die Ausgänge mit Output-Enable auf den gepeicherten Wert geschaltet ($\overline{OE}$ low) oder auf hohe Impedanz ($\overline{OE}$ high) geschaltet werden.
Intern handelt es sich um CMOS static RAM (6116).



Mode	$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	I/O
Standby	H	X	X	Isolation
Read	L	L	H	Data out
Read	L	H	H	Isolation
Write	L	X	L	Data in

Changelog

Table 7: Changelog

Datum	Version	Änderung
04.12.2011	0.9	Initiale Version, es fehlt noch die Beschreibung von 74LS245
05.12.2011	1.0	Beschreibung von 74LS245
05.12.2011	1.01	Korrektur Rechtschreibfehler
05.12.2011	1.1	Einfügen Übersichtsplan mit Programmierlogik
08.12.2011	1.2	Einfügen Beschreibung RAM; Hinweis auf Flipflop für Carry-Bit im Akku; Korrektur im Programmzähler (m0, m1-> s0, s1)
09.12.2011	1.3	Korrektur Statusregister Zero-Bit; Korrektur Belegung ALU; Umbenennung der Steuerleitungen s0, s1 in k0, k1; Ersetzen Instruktionszähler durch Programmzähler; Korrektur Steuerleitungstabelle Instruktionszähler
10.12.2011	1.4	Hilfestellung zum Erzeugen des Taktsignals, Hinweis zum Speichern des Carry-Bit in einer Erweiterung des Akkumulatorregisters
03.11.2014	1.6	Texte + Layout überarbeitet, für WS14/15, Vorbereitung und Testfälle besser ausgearbeitet
25.11.2014	1.7	Texte überarbeitet; Programmierlogik ausgetauscht (mit 4-bit-Adressen); Übersichtsplan mit Aufteilung auf Digiboards; Bildunterschriften; Tippfehler beseitigt
10.12.2014	1.8	Instruktionsdekoder Leitung *ind:* Polarität korrigiert
11.12.2014	1.9	Instruktionsdekoder Leitung *ind:* Polarität wieder der Logik angepasst; Steuerwerk: Signal *m1* wird zum invertierten *ind* Signal
11.12.2014	1.10	Anpassung ALU-Funktionstabelle; Text beim RAM
30.11.2015	1.11	Parallele Phasen: Execute und Memory Access; Update von Titeln
01.12.2016	1.12	Anpassung Diagramme; Testprogramme in die Vorbereitung aufnehmen
23.12.2016	1.13	Änderung der Tasterverzögerungsschaltung; Nullkraftsockel-Nummerierung; Hinweis auf Fehler der ALU-Beschreibung in der Digiboarddokumentation; Hinweis zu Problembehebungen bei schlechtem CLK-Signal; Testanweisung Fremdtesten eingefügt
18.01.2017	1.14	Bei indirekten Befehlen () hinzugefügt
30.11.2018	2.0	Konsolidierung
07.12.2018	2.1	Bei Erklärung der indirekten Befehlen () hinzugefügt
04.11.2021	2.2	Update für Corona-Mode des Praktikums: Bilder 10 und 17 aktualisiert, in Bild 18 ProgrammingEnable Logik auf High Active geändert
08.11.2021	2.3	Anpassung der bedingten Sprungbefehle an die Syntax im Buch (d.h. BRx ~n → BRx #n)
22.10.2022	3.0	Überführung in Markdown und Korrektur
16.11.2022	3.1	Schreibweisen harmonisieren