DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Der von-Neumann und der MU0-Modellrechner

DHBW Duale Hochschule Baden-Württemberg Heidenheim

Ein kleines Programm…

Hochsprache (C, C++, C#...): var += 2;

↓ Compiler

Assemblersprache: → Assembler → Maschinencode (16-Bit-Befehle):

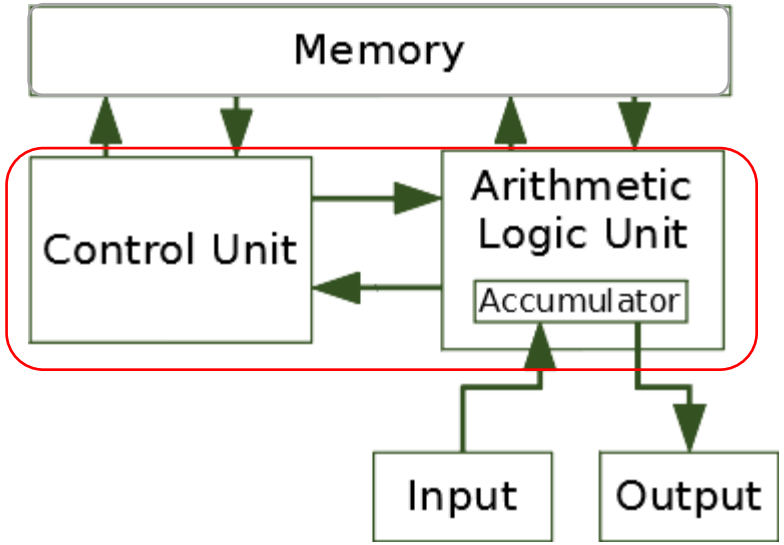
		Assembler	Maschinencode
	LDA var	→	0000.ssss|ssss.ss00
	ADD inc	→	0010.ssss|ssss.ss10
	STO var	→	0001.ssss|ssss.ss00
ssss.ssss.ss00	var: DEFW #0	@ hier: zunaechst unbelegt	
ssss.ssss.ss10	inc: DEFW #2		

Befehlscode + Operand(en)

Der „von Neumann“-Rechner (Speicherprogrammierter Rechner)

Programmcode und Daten werden gleichbehandelt

CPU



Veröffentlicht 1945 in „First Draft of a Report on the EDVAC“

Arithmetic Logic Unit - Rechenwerk

- führt Berechnungen durch

Control Unit - Steuerwerk

- interpretiert die Anweisungen und steuert die Befehlsabfolge

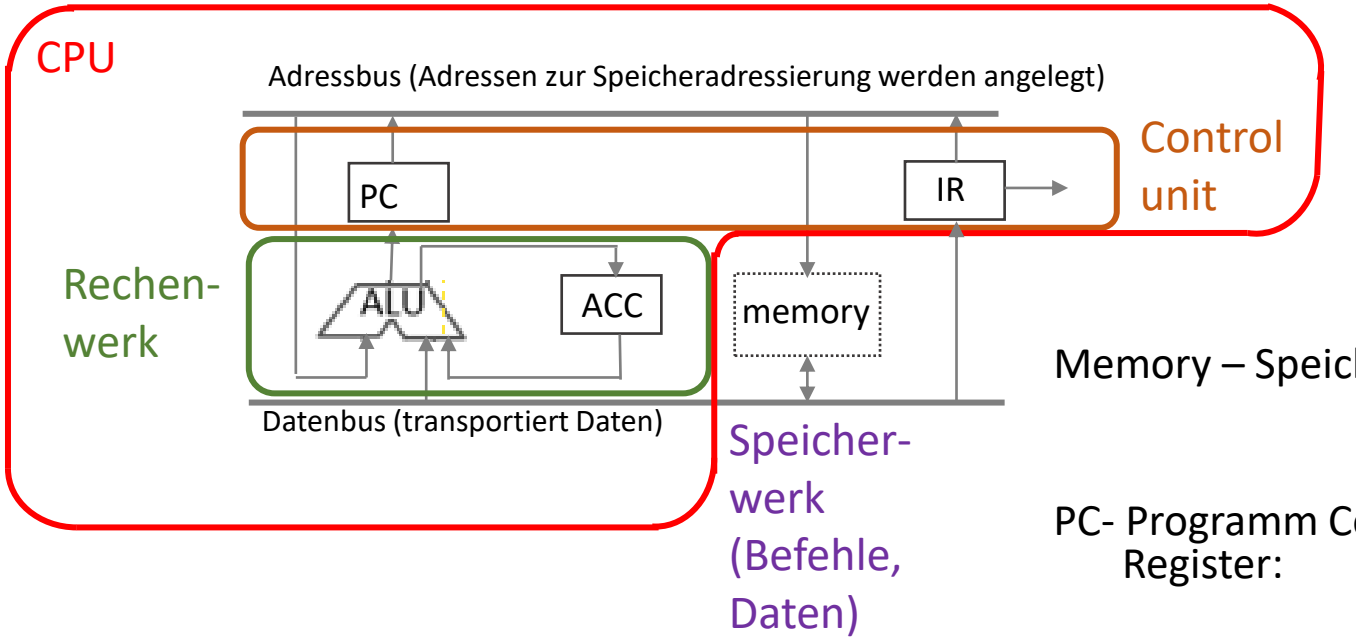
Memory - Speicherwerk

- speichert sowohl Programme als auch Daten (JvN.)

Input/Output - Ein/Ausgabewerk

- steuert die Eingabe und Ausgabe von Daten zum Anwender

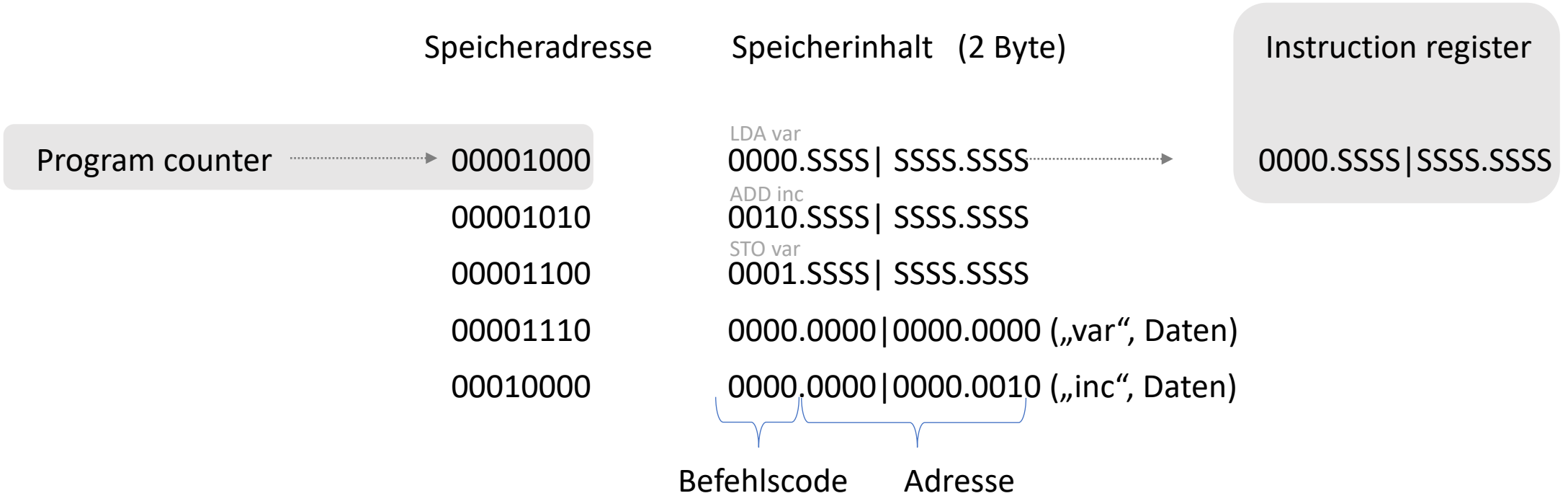
Die Elemente eines einfachen Rechners inkl. Busse und Speicherinhalt



Nicht dargestellt: Ein-/ Ausgabewerk

Memory – Speicher:	Speichert Befehle (Programm) und Daten (Operanden)
PC- Programm Counter Register:	Speichert die nächste Programm- (Befehls-) adresse
IR - Instruction Register:	Speichert den aktuellen Befehl
Bus:	Verbindet alle Elemente
Control unit:	Steuerwerk, Takt
ALU- Arithmetic Logic Unit:	kann mit Operanden rechnen
ACC- Akkumulator Register:	Ergebnisregister für Berechnungen

Unser kleines Programm…



Lesen und schreiben von Daten, Speicher und Registern

Blickrichtung des Datentransfers ist immer von der CPU (vom Steuerwerk/ von den Registern) aus, Richtung Speicher.

Steuerwerk/ Register (schnelle Zwischenspeicher) sind die aktiv Handelnden, Speicher ist passiv!

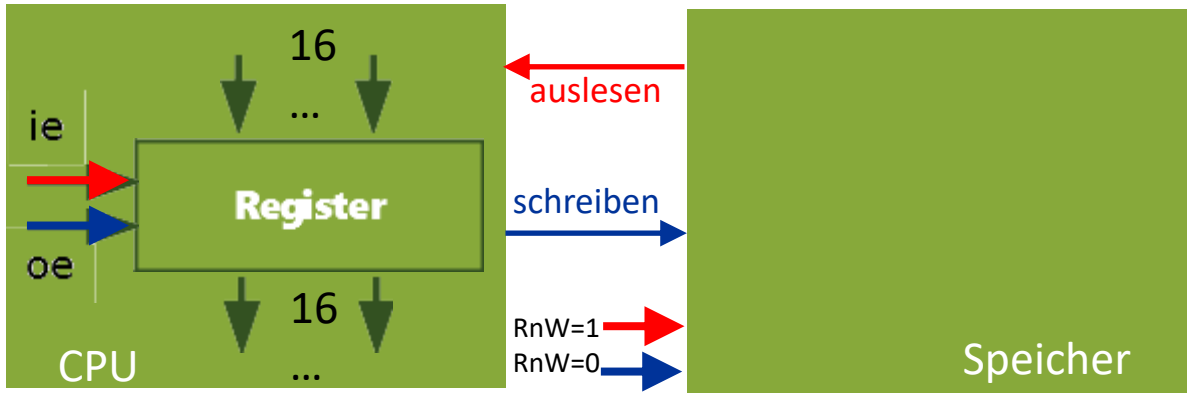
- Signal ie (input enable) liegt an: Register lesen ein (aktiv!)
- Signal oe (ouput enable) liegt an: Register schreiben (aktiv!)

Aber:

- Speicher wird beschrieben (passiv)
- Speicher wird ausgelesen (passiv)

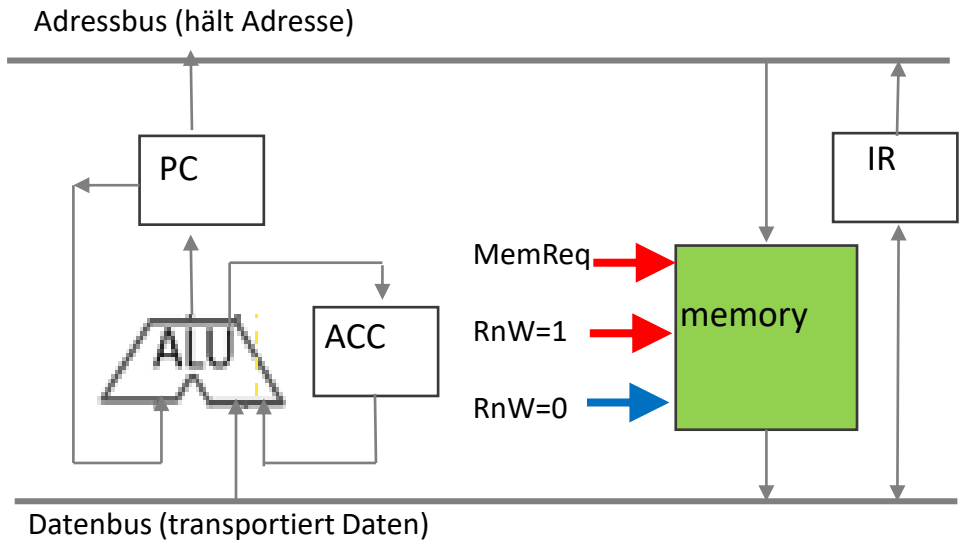
Farbcode:

Schreiben
Lesen



Hauptspeicher

Schreiben
Lesen



Speicherstellen werden über Adressen angesprochen, um die Daten auszulesen

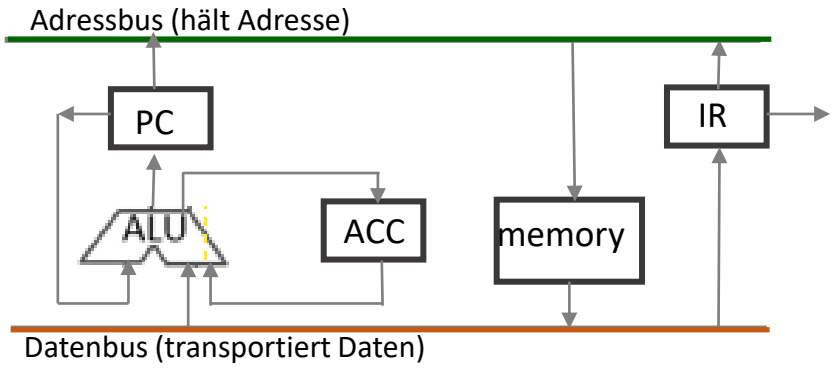
(Adresse auf Adressbus schreiben -> Daten vom Datenbus lesen)

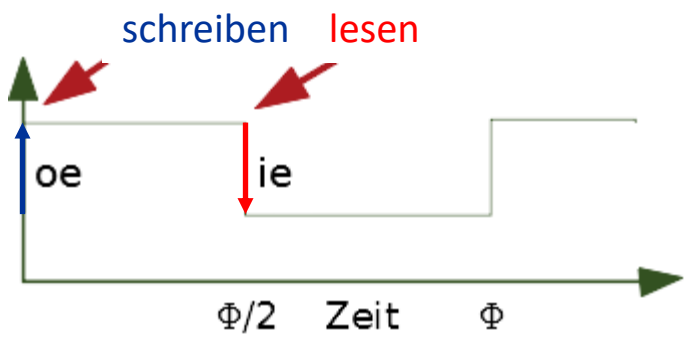
Adresse	*Daten*
0xn	Wort 1
0xn + 4	Wort 2
0xn + 8	Wort 3
0xn + 12	Wort 4
	…

Breite des Adressbusses: entsprechend Speicherelementzahl
(1024 Speicherworte: ?)

Breite Datenbus: entsprechend Wortbreite (32 Bit bei 32-Bit-Daten)

Adress- und Datenbus





Je nach Steuercode des Steuerwerks, lesen oder schreiben die Komponenten (Adressen oder Daten) am Bus.

Die jeweiligen Daten liegen am gesamten Adress- bzw. Datenbus an.

Es darf pro Bus immer nur 1 Element geschrieben werden.

Entsprechend der Taktung kann in einem Takt auf den Bus **geschrieben und gelesen** werden.

Heisst: Bis hierhin kann in einem Takt ein Register nicht Werte einlesen und wieder ausgeben!

Adressbus: Hält Adressen

Datenbus: Transportiert Daten

Datenbus-Breite = 1 Daten-Wort

Welches Register enthält nur Adressen?

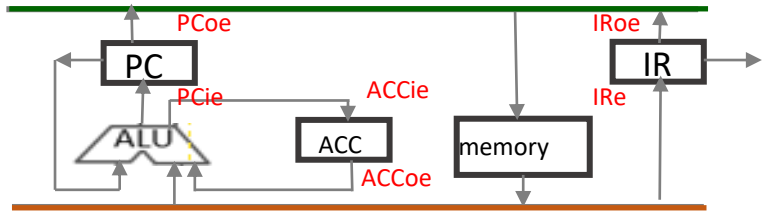
Steuerwerk/ Control unit (Befehlsabarbeitung)

Besteht aus:

- Program counter (PC)
- Instruction register (IR)
- Befehlsdecoder
- Befehlsablaufsteuerung durch Anlegen von Steuer-signalen an die Komponenten

Vorbedingung

Inputs						**Outputs**																**Funktion**
Instruction	Opcode	/Reset	step	Z	N	step	Acress	ACCce	ACCie	PCce	PCie	IRce	IRie	SPce	SPie	DINce	DINie	DCLTce	DCLTie	ALU Function	MEMrq	R/rW	
Fetch	xxxx	1	0	x	x	1	0	0	0	1	1	0	1	0	0	0	0	0	0	A+1	1	1	IR=[PC], PC=PC+1

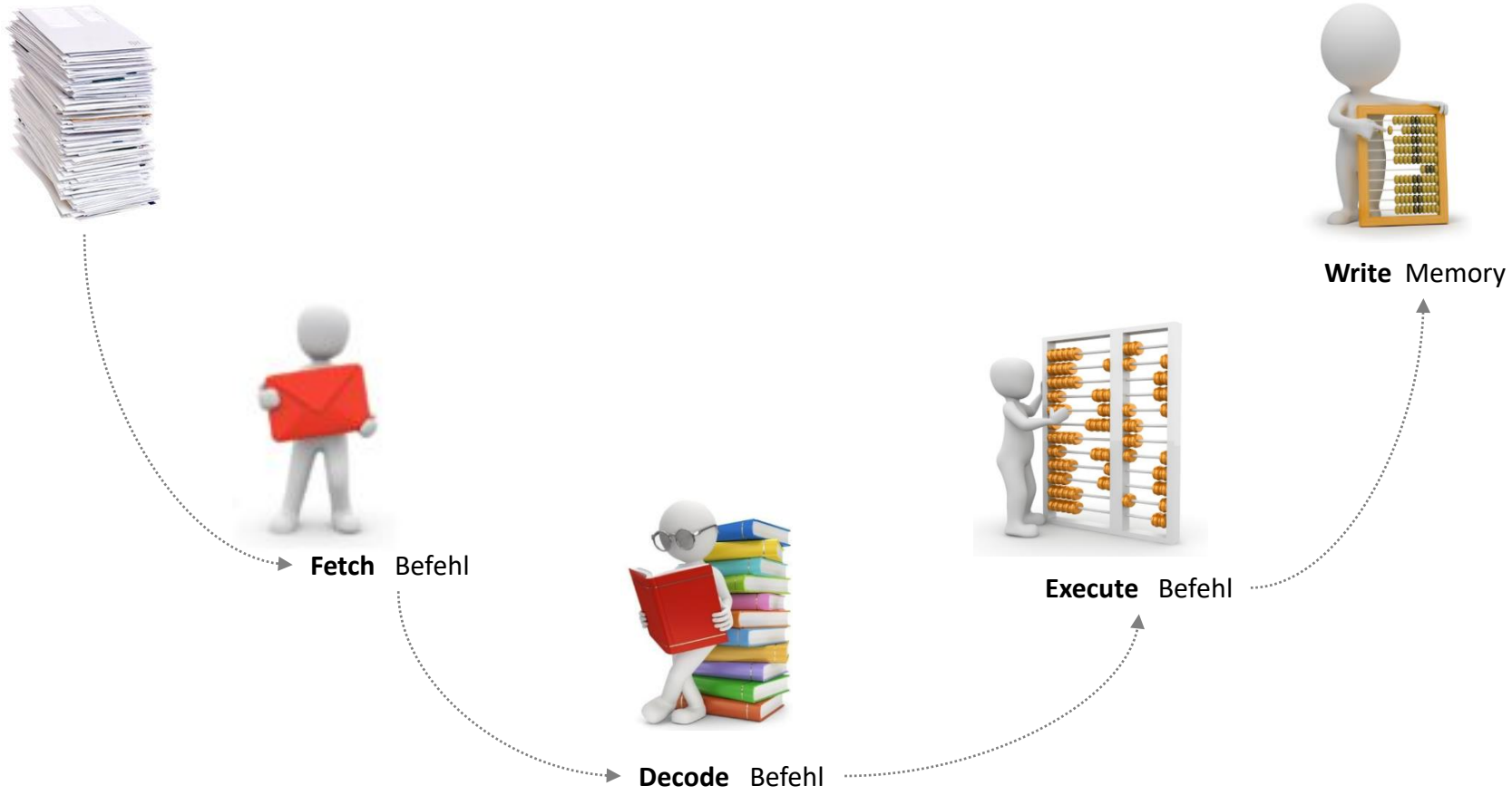


Steuert den **Ablauf der Befehlsbearbeitung** durch Steuersignale zu den Komponenten über den Steuerbus.

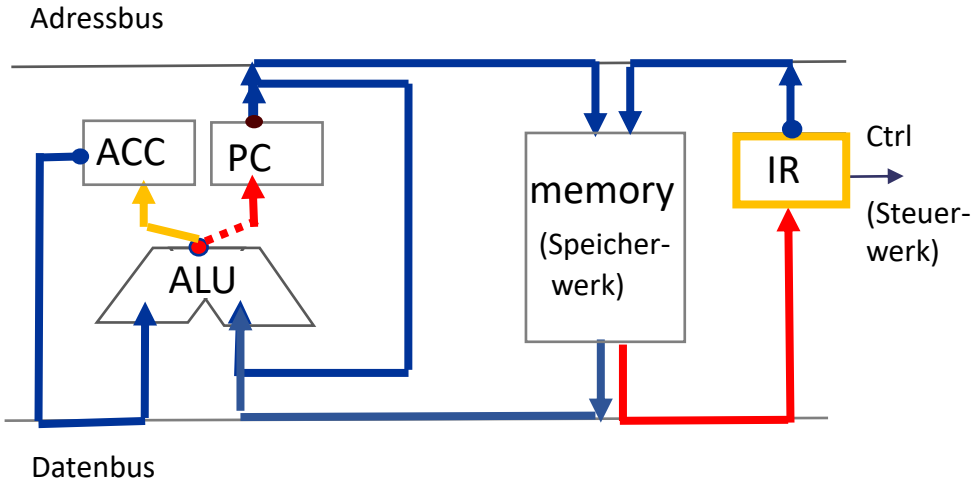
Alle Aktivitäten des Steuerwerks folgen dem Takt der CPU.

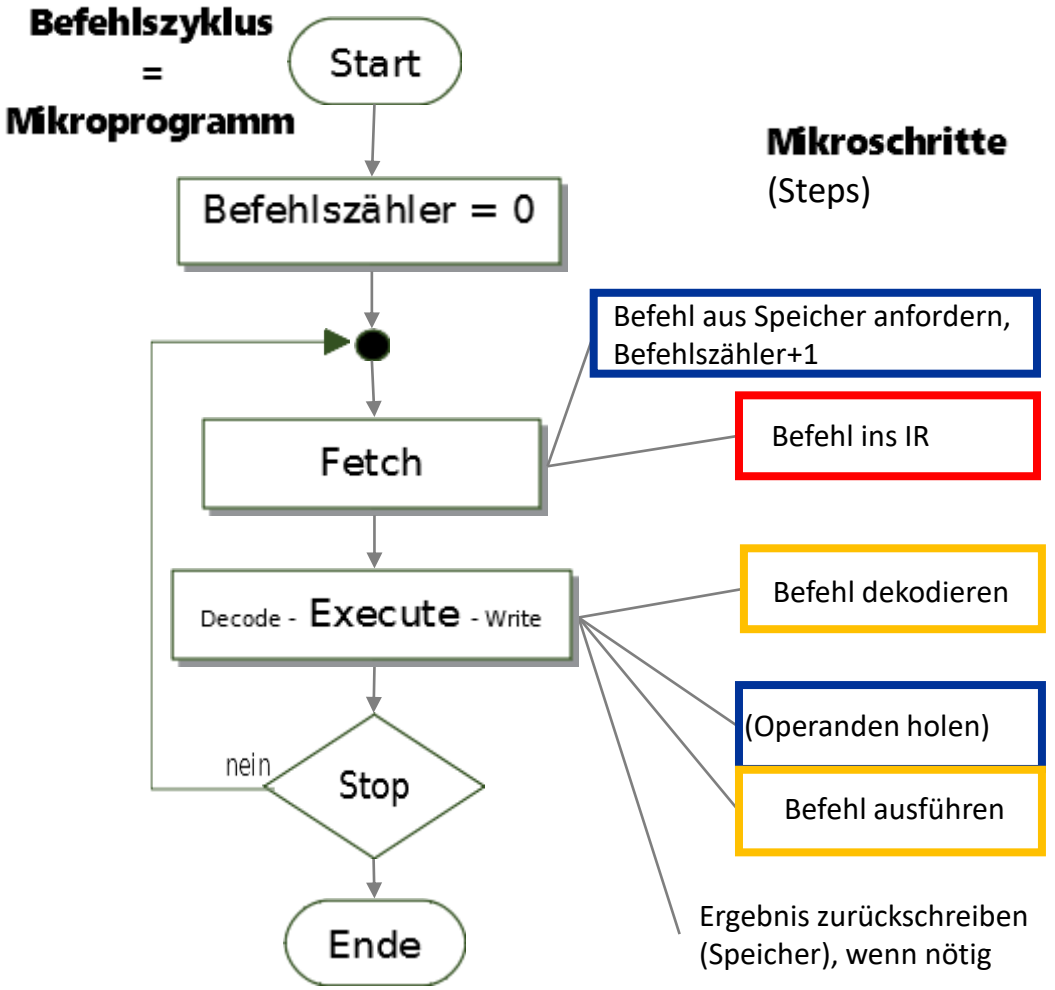
Das Steuerwerk führt für jeden Befehl ein Mikroprogramm, bestehend aus Mikroschritten (Steps), aus.

Befehlsablauf (z.B.: LDA #1 – Lade Akku mit 1)

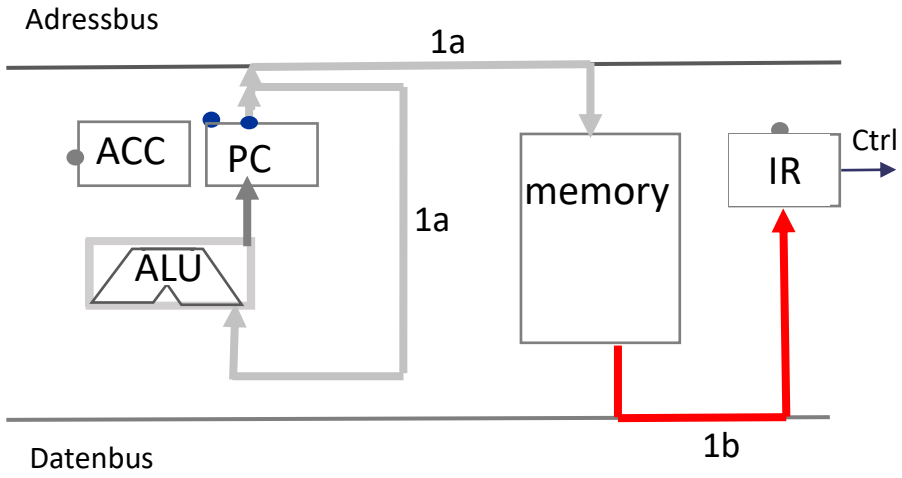


Beispiel für einen Befehlsablauf (ADD s)





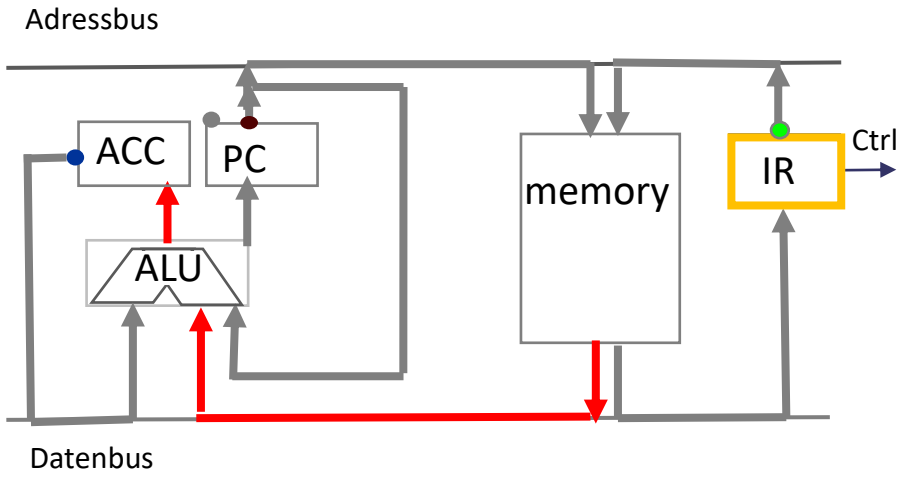
Der Befehlszyklus: Fetch



- Im Fetchzyklus wird der Wert des Programcounters auf den Adressbus gelegt und der Programmcode aus dem Speicher in das Instructionregister gelesen
- Die ALU erhöht gleichzeitig den Wert des Programmcounters und speichert diesen wieder zurück
- Benötigt einen vollen Taktzyklus durch Lesen des OpCodes aus dem Speicher

- Blau: die Register schreiben (oe)
- Rot: die Register lesen (ie)

Der Befehlszyklus: Decode/ Execute/ Write für Rechenoperationen

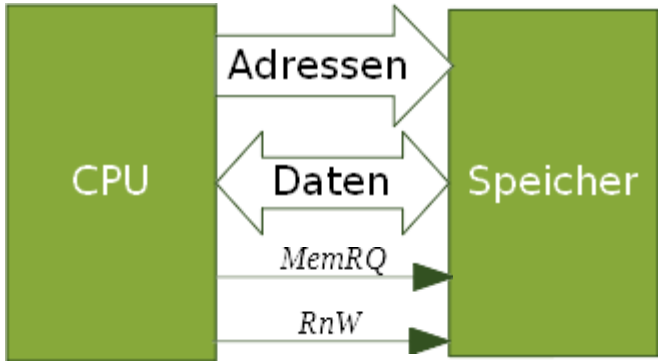


Bei einer Rechenoperation wird:

- die Adresse des Operanden aus dem Befehl dekodiert und
- der Wert des angesprochenen Speicherworts an den einen Eingang der ALU geschaltet (Speicherzugriff: 1 Takt!).
- Der zweite Eingang der ALU erhält seine Information vom Akkumulator
- Das Ergebnis der Rechnung wird wieder im Akkumulator gespeichert.

Speicher

Ein Speicher besteht aus einer Menge von Speicherworten, die durch eine Adresse anwählbar sind



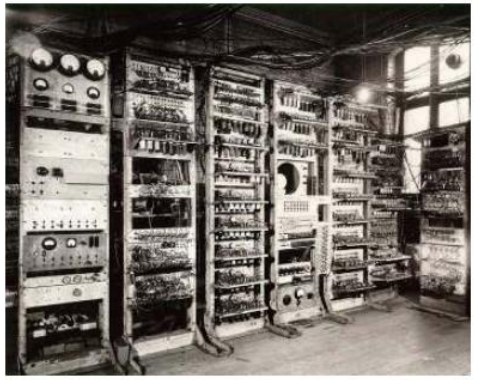
Steuersignale des Datenbusses:

MemRQ (Memory Request): Der Speicherbaustein wird angesprochen. Dieses Signal ist notwendig, da mehrere Speicherbausteine vorhanden sein können.

RnW (Read / not Write): Dieses Signal gibt an, ob von dem Speicher gelesen wird, oder ob eine Information in den Speicher geschrieben wird. Wenn nicht geschrieben wird, sollte RnW immer auf Read geschaltet sein

R/nW: Lesen bei High-Pegel, Schreiben bei L-Pegel
(ein führendes „n“ im Signalnamen bedeutet active:low (0=aktiv))

Historie des MU



Dimensions

Length: 5.23 m (17 ft)

Height: 2.26 m (7 ft 4 in)

Weight: 1 tonne

- Der MU-Rechner ist nach dem ersten funktionierenden Digitalcomputer der Welt, dem SSEM (Small Scale Experimental Machine) der Universität Manchester modelliert worden
- Der SSEM führte am 21. Juni 1948 zum ersten Mal ein Programm aus dem Speicher heraus aus
- Es ist damit die erste erfolgreiche Implementierung eines von Neumann Rechners
- Er wird an der Universität Manchester zu Lehrzwecken eingesetzt

→ Wir nutzen ihn als Modellgrundlage für eine Lernreise durch die Prozessorkonzepte

→ Manchester Baby:

https://en.wikipedia.org/wiki/Manchester_Baby

- weiterentwickelt zu (Ferranti) Mark I, weltweit erster kommerziell vertriebener Allzweck-Computer

Historie des MU

Wir werden gemeinsam folgende Modell-Rechner-Generationen entwickeln:

MU0:	Basiskonzept, Befehlsabarbeitung
MU1:	Adressierungsmodi, Akkumulator-Architektur
MU2 & MU3:	Adressraum vergrößern
MU4:	Registeranzahl erhöhen
MU5:	Adressberechnung, Register-Architektur, Load-Store
MU6:	Harvard-Design
MU7:	Pipelining, „ARM-ähnlich“

Umsetzung des MU0 in Hardware

Im Folgenden:

- Datenpfad Modell: Fokussierung auf die Datenflüsse zwischen den Komponenten
- Steuerung der Datenflüsse durch Steuercodes (Steuerwerk)
- Taktsynchronität (zeitliche Abfolge)
- Codierung von Befehlen: Opcodes
- Mikroprogrammierung für komplexe Befehle

Modellrechner-Generation MU0: Annahmen

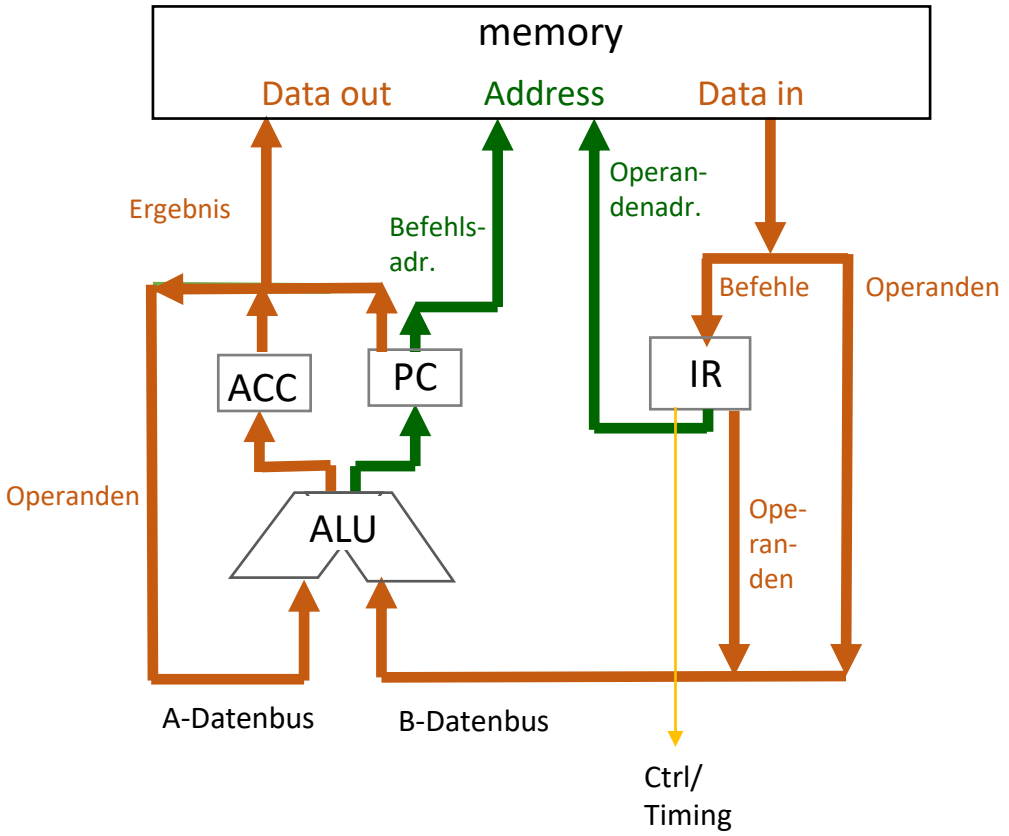
- Hauptspeicherzugriffe sind sehr langsam (das Bottleneck):

  jeder lesende Speicherzugriff benötigt einen vollen Taktzyklus!

- Heißt für MU0: jeder Befehl braucht mindestens soviel Taktzyklen, wie er Speicherzugriffe braucht

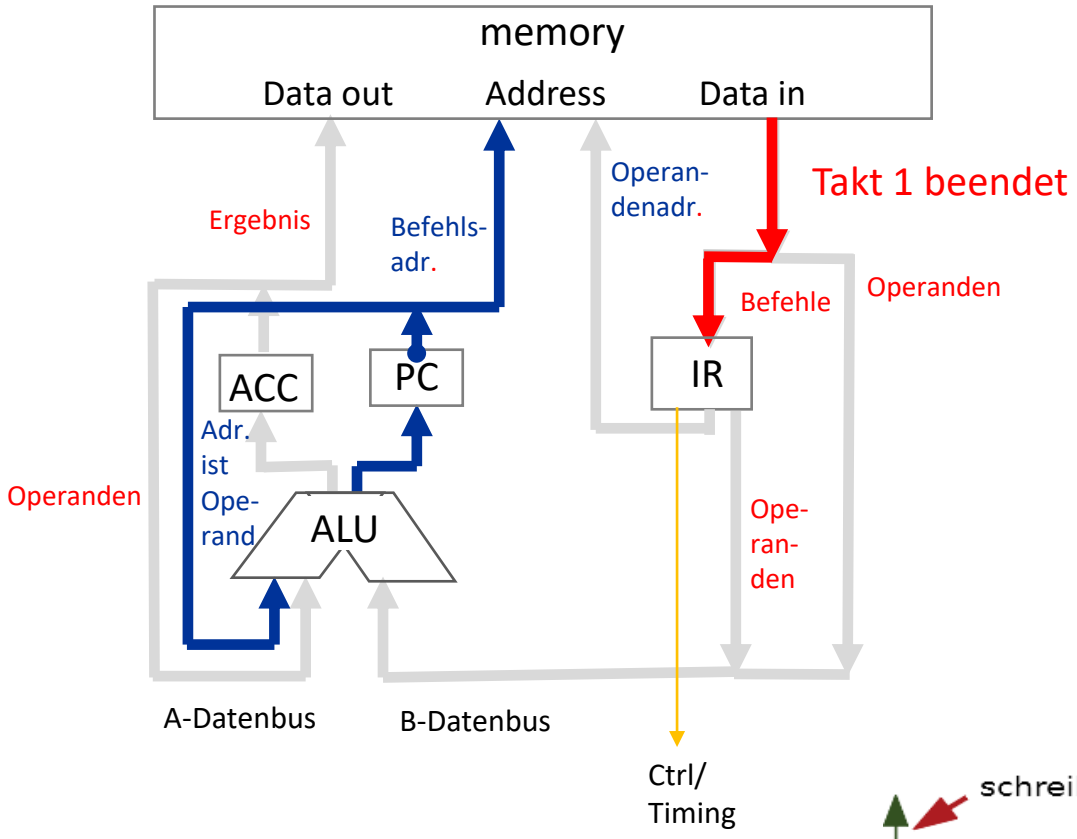
Fokussierung auf Prozessor (Verschaltung der Register)

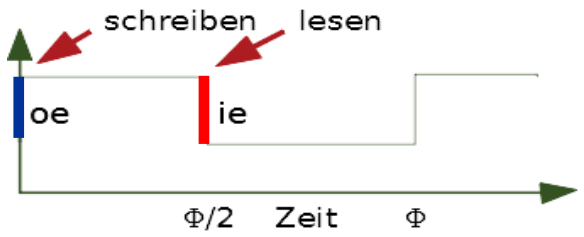
- Memory (Speicher) ist nicht Bestandteil des Prozessors
- Andere Darstellung für Datenfluss durch Verschaltung der Komponenten, Trennung in Adress- und Datenbus ist dafür weniger relevant
- Zwei Datenbusse zu den Eingängen der ALU, A und B



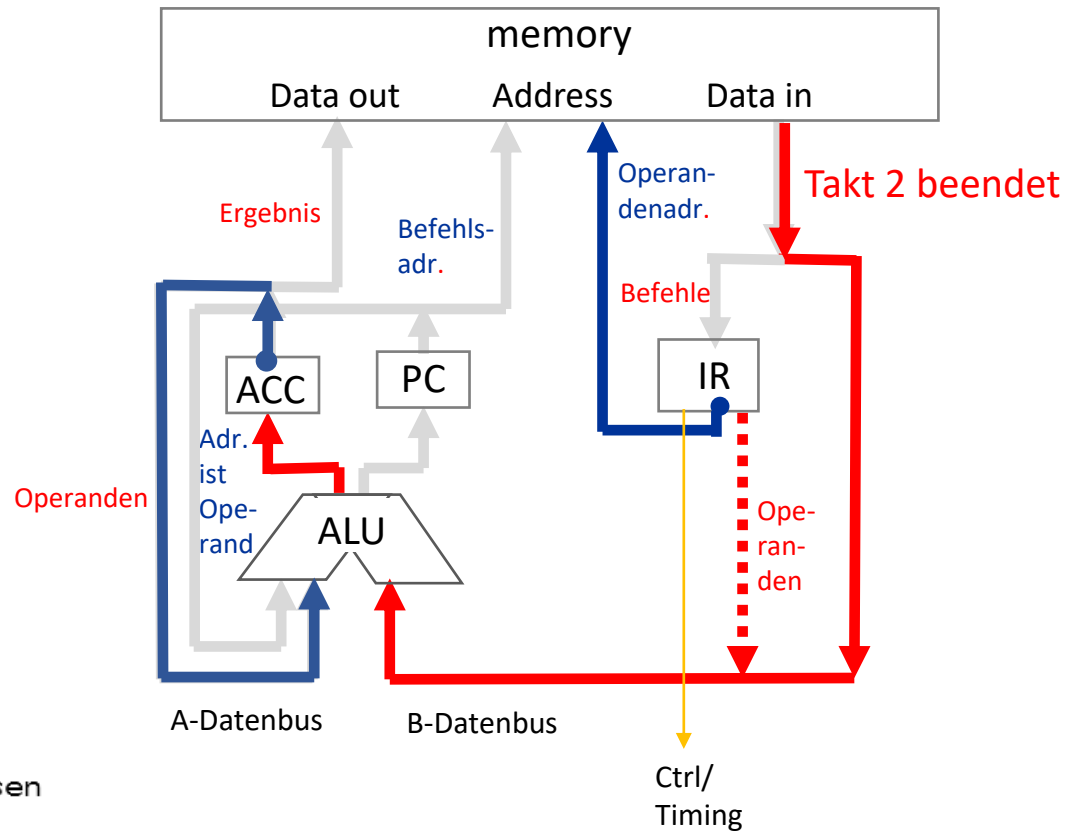
Befehl ADD s

Fetch

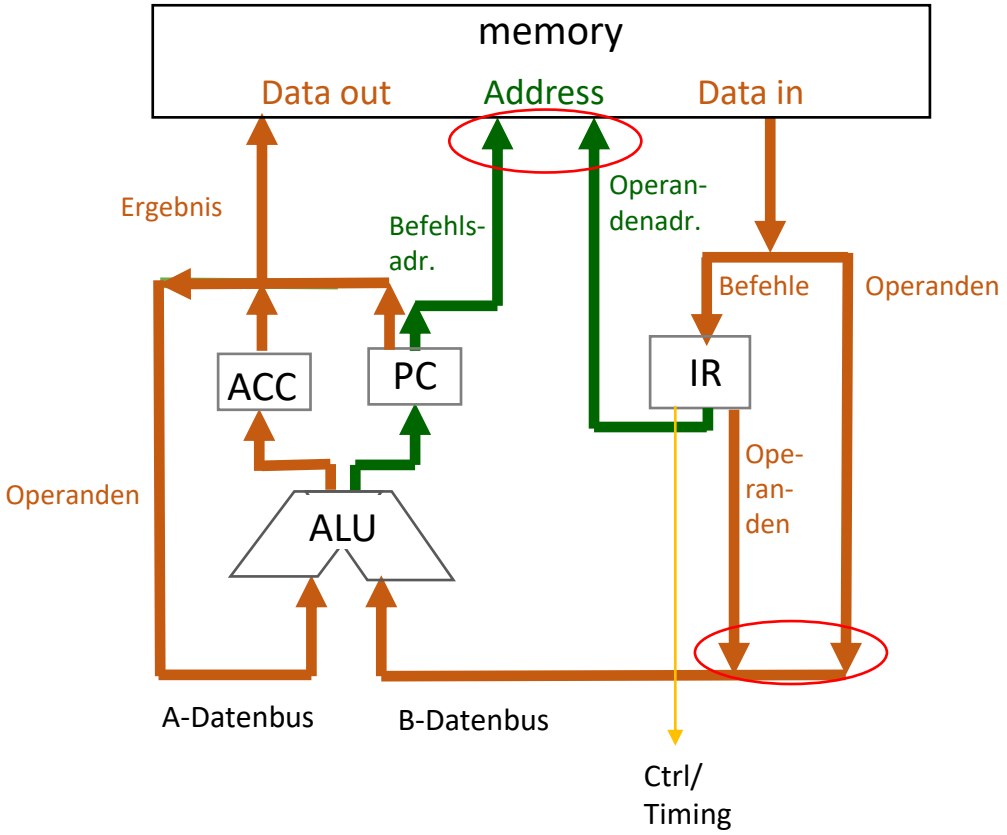




Execute

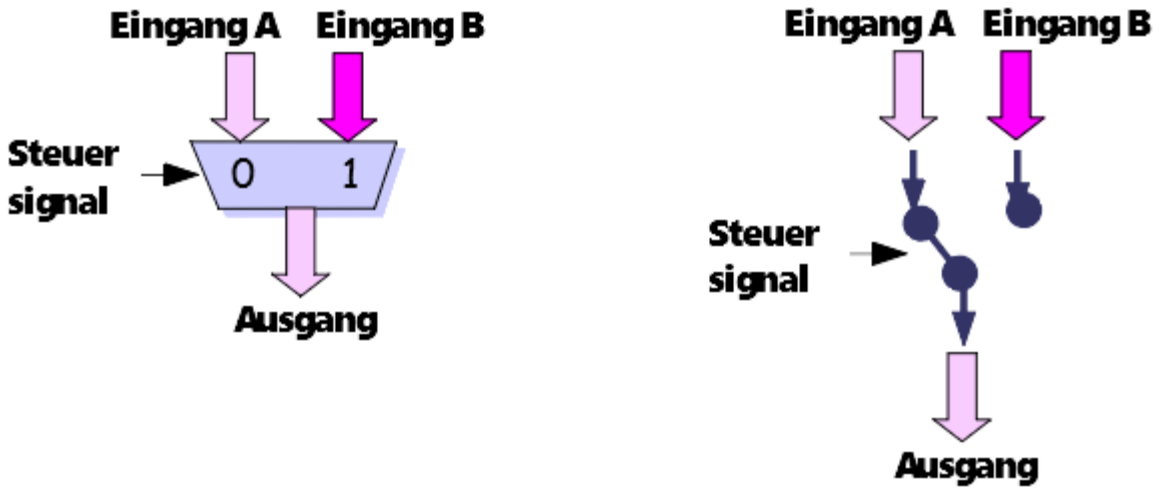


Wer darf wann auf den Bus schreiben?



Der Multiplexer

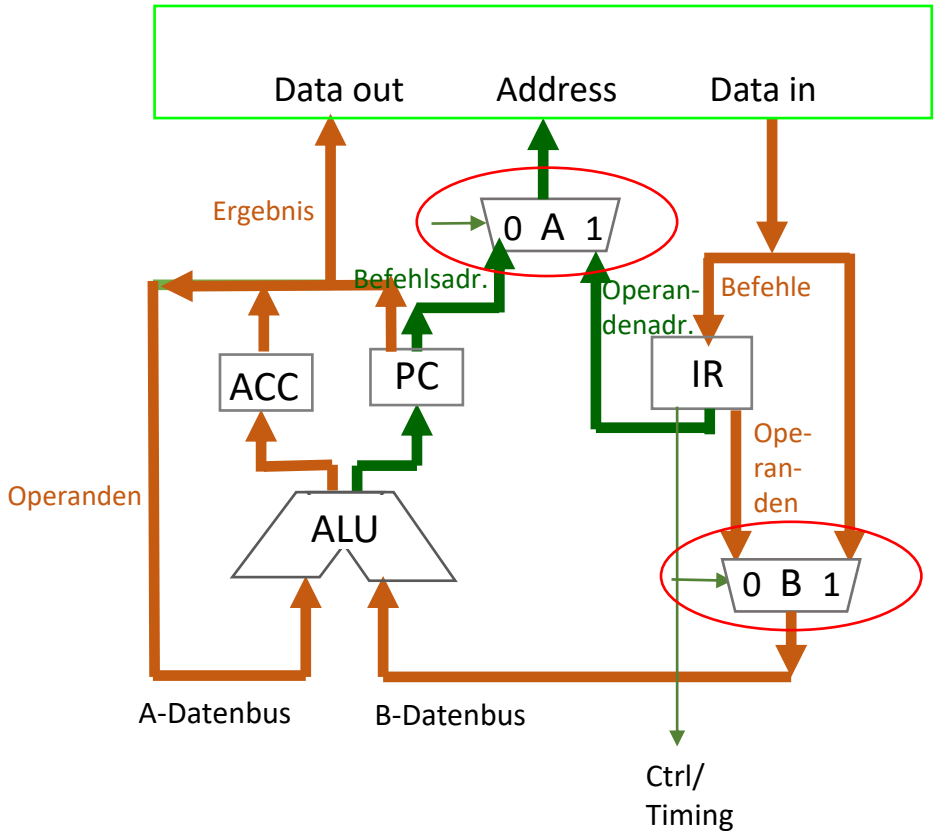
Der Multiplexer ist ein **Schalter**, der durch das Anlegen eines Steuersignals das ausgewählte Eingangssignal auf den Ausgang legt



Das modifizierte MU0 Datenpfad Modell

- Die ALU hat zwei Eingänge, A und B
- Kontrolle des Datenpfades und Adressbusses durch Multiplexer

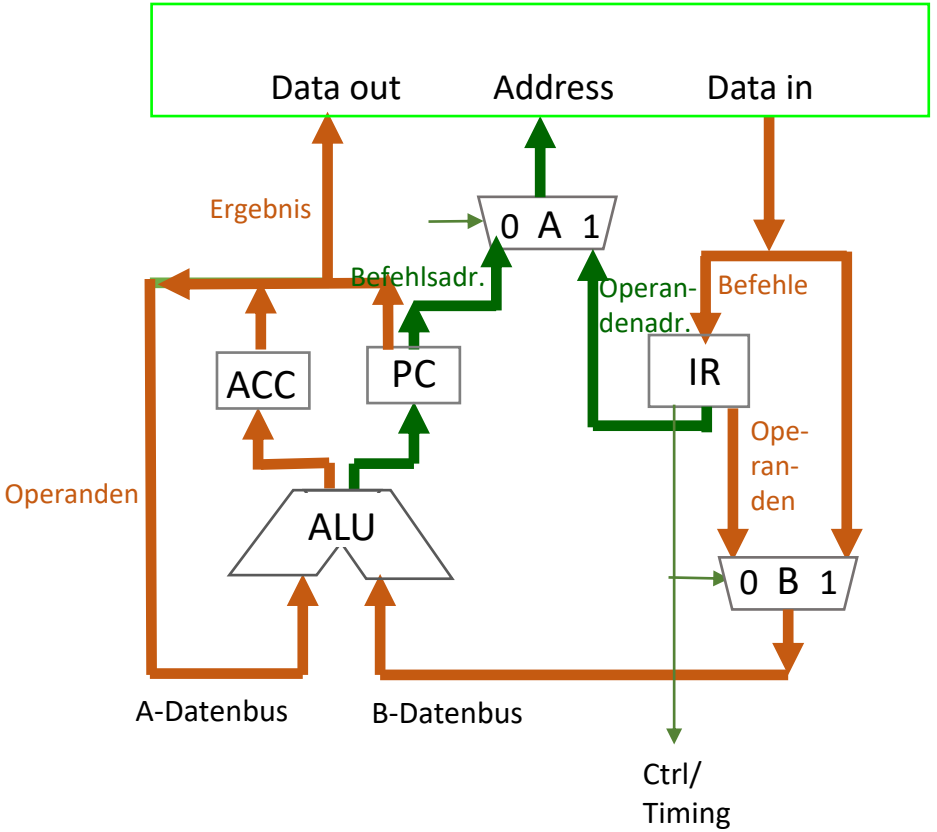
Durch diese Modifikation ist eine Realisierung des MU0 mit max. 2 Takten pro Befehl möglich



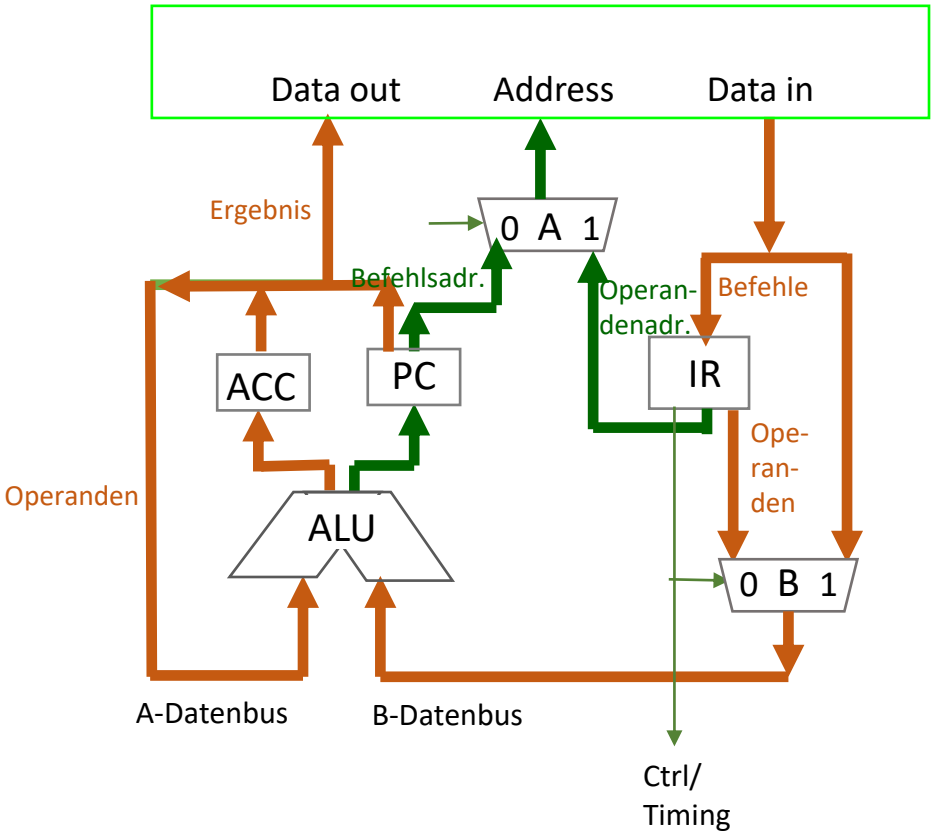
Das modifizierte MU0 Datenpfad Modell

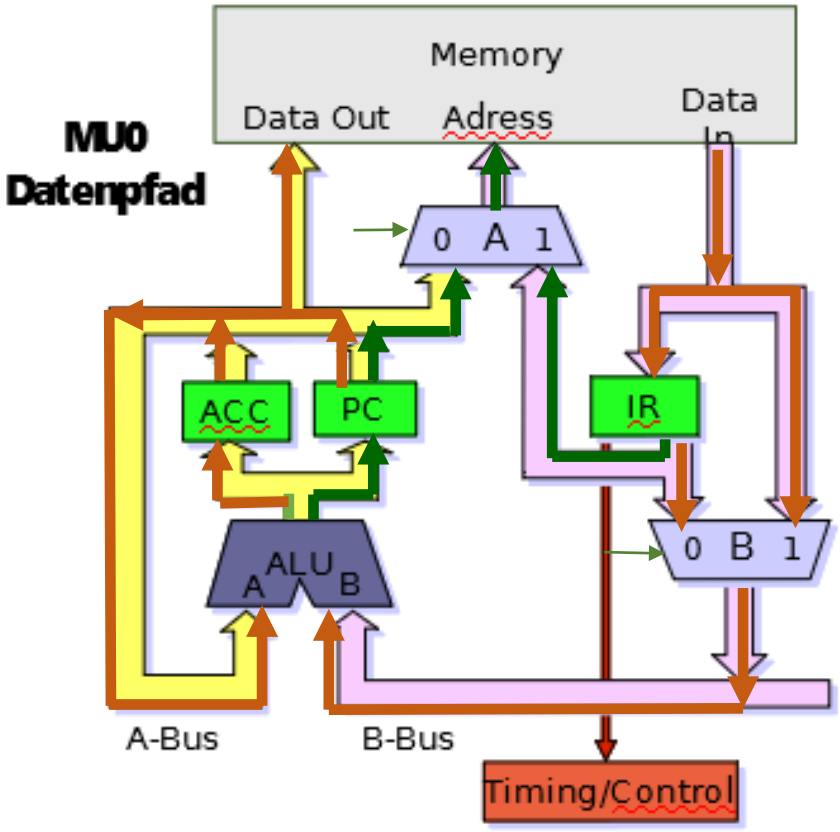
- Der Multiplexer B steht:
  - auf 1 bei allen Datenverarbeitungsbefehlen und
  - auf 0 bei allen Sprungbefehlen. Bei Sprungbefehlen erfolgt kein Datenzugriff, S (IR) wird direkt nach PC verschoben

- Der Multiplexer A wird bei den bisherigen Befehlen nur für den Fetchzyklus auf 0 gestellt, ansonsten wird S (unterer Teil von IR) immer als Adresse interpretiert



Adressbereitstellung und Datentransport werden gemeinsam als Datenpfad dargestellt





Takt und Steuerung MU0

- 1 Instruction

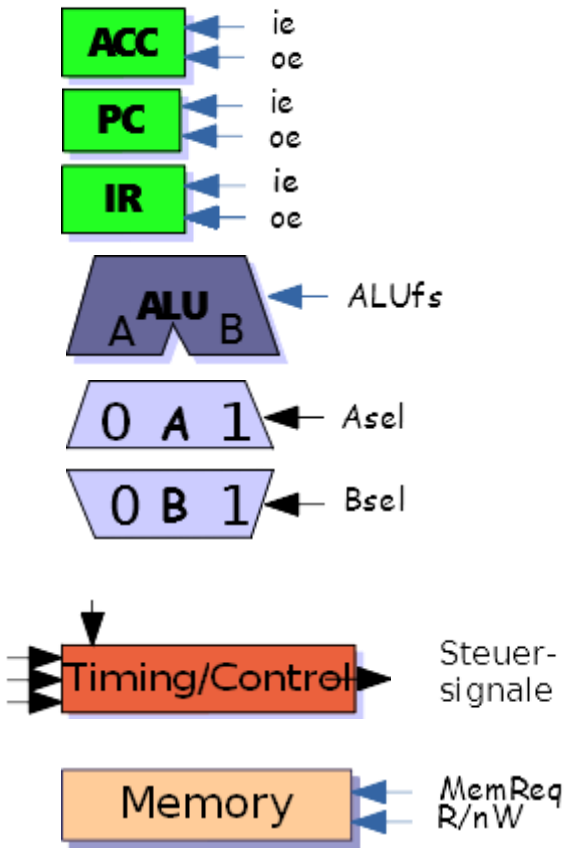
  = 2 Takte Fetch + Execute

  = 4 Taktflanken Schreiben + Lesen + Schreiben + Lesen

- Steuerung der Elemente erfolgt durch Steuercodes: ie, oe

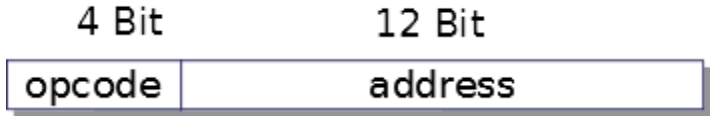
- Wortbreite des Prozessors: 16 Bit
  - Registergröße
  - Datenbusbreite
  - Eingänge/Ausgänge ALU

MU0 Elemente und ihre Steuercodes



Akkumulator:	ie -> Lesen, oe -> Schreiben
Program Counter:	ie ->Lesen, oe -> Schreiben
Instruction Register:	ie -> Lesen, oe → Schreiben
ALU:	Funktionsauswahl über ALUfs (function select)
Adressbusmultiplexer A:	0 -> PC, 1 -> Adressteil von IR
Datenbusmultiplexer B:	0 -> Sprungadr. von IR oder konstanter Operand 1 -> Daten aus Speicher
Steuerwerk:	Erzeugt die Steuersignale aus IR, ALU Status, Reset und internem Speicher
Speicher:	MemReq -> Speicher wird angesprochen, R/nW -> 1 Lesen, 0 Schreiben

Instruction Set des MU0 Rechners



Die Instruktionen setzen sich zusammen aus

- einem 4 Bit Operation Code und
- einer 12 Bit Adresse

Der Mu0-Prozessor kennt 8 (von 16 möglichen) Befehlen

Instruction	Opcode	Effect
LDA s	0000	ACC = mem[s]
STO s	0001	mem[s] = ACC
ADD s	0010	ACC = ACC + mem[s]
SUB s	0011	ACC = ACC – mem[s]
JMP s	0100	PC = s
JGE s	0101	IF ACC >= 0 PC=s
JNE s	0110	IF ACC != 0 PC=s
STP	0111	stop

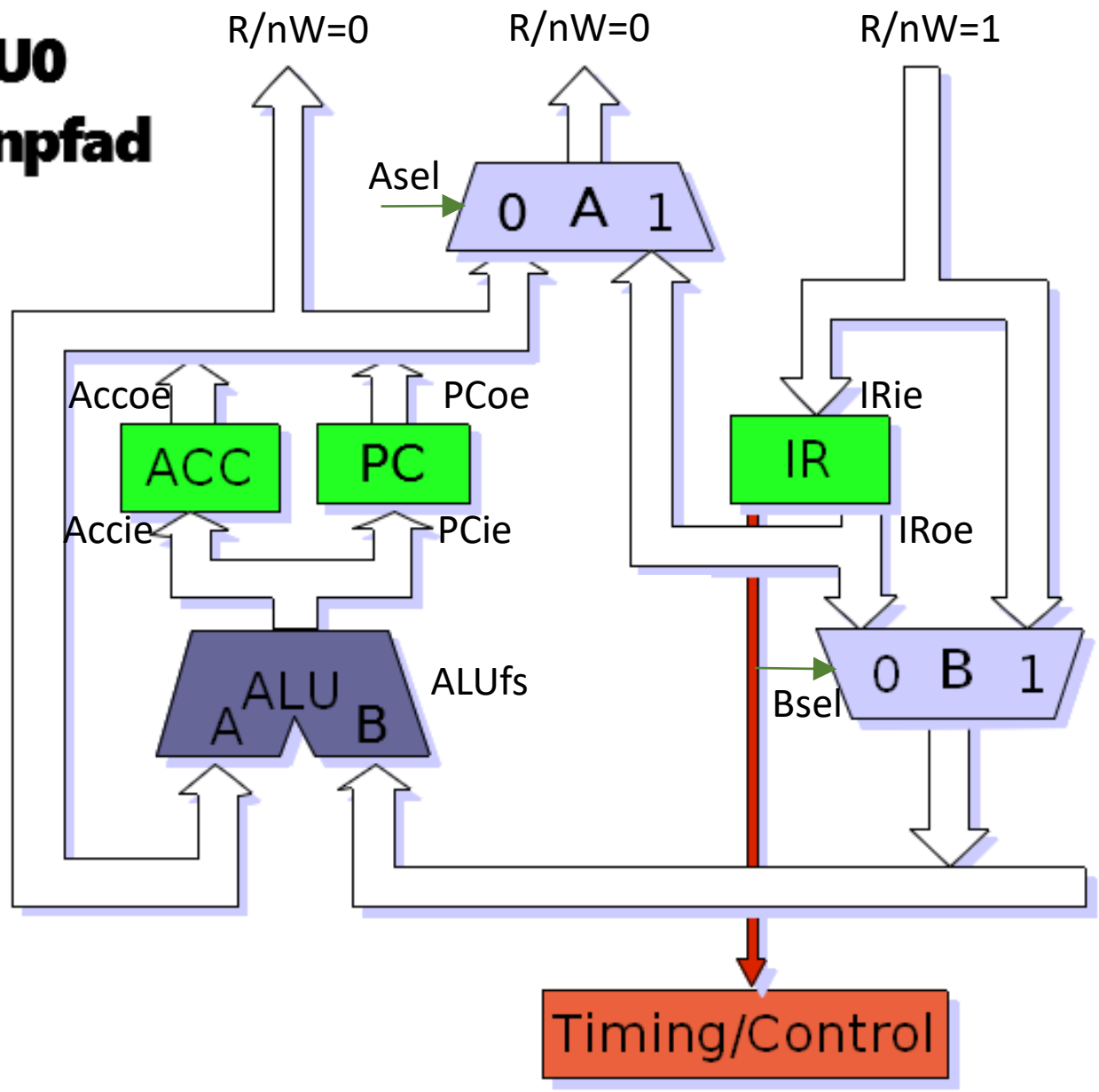
Load <s>
Store <s>

Add <s>
Sub <s>

Jump
Jump Grt./Equ.
Jump Not Equ.
Stop

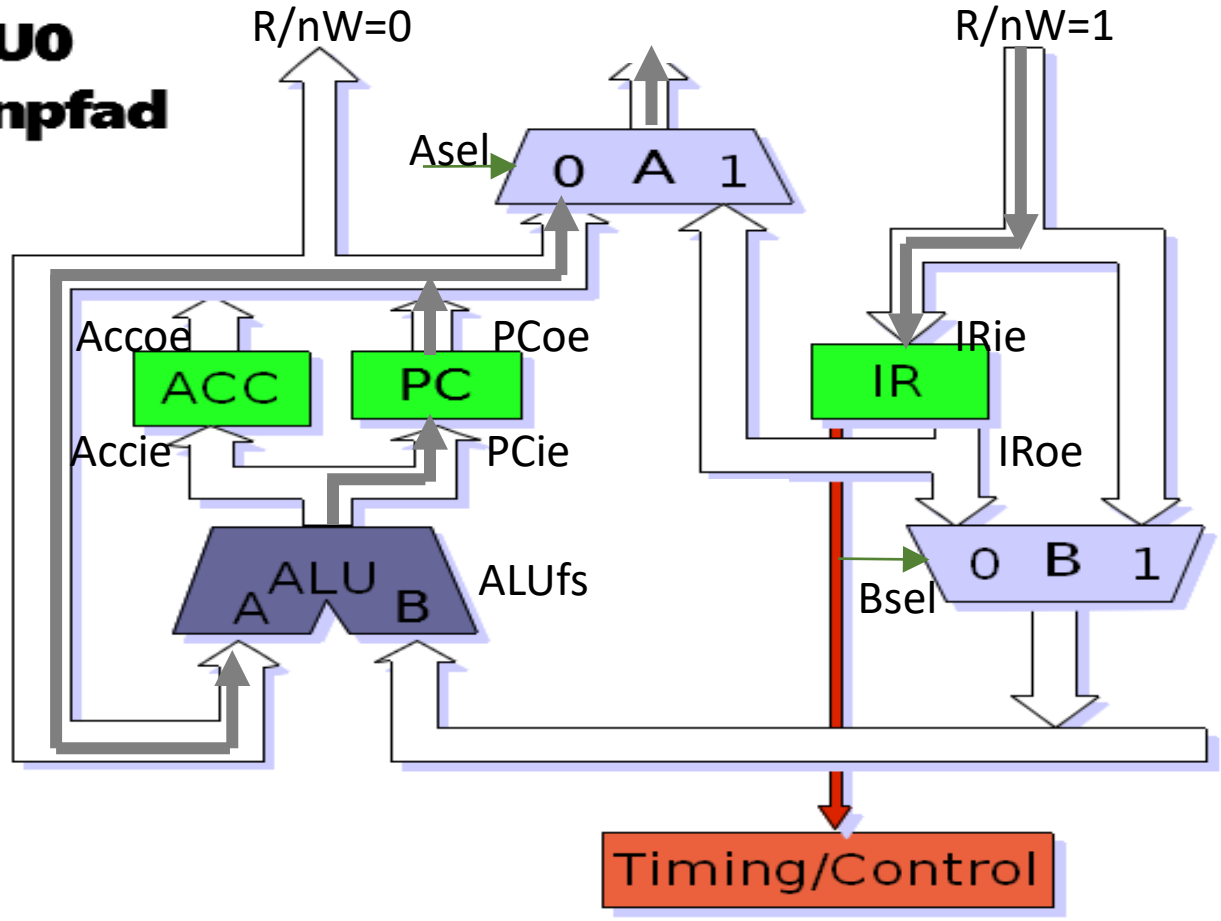
- zum Laden und Speichern von Werten
- zur Berechnung und
- zur Ablaufkontrolle

MJ0 Datenpfad



MJ0
Datenpfad

Steuertabelle



Inputs						Outputs											
Instruction	**OpCode**	**Reset**	**Step**	**ACCz**	**ACC15**	**Step**	**Asel**	**Bsel**	**ACCoe**	**ACCie**	**PCoe**	**PCie**	**IRoe**	**IRie**	**ALUfs**	**MEMrq**	**R/nW**
Reset	XXXX	1	x	x	x	0	x	x	0	0	0	1	0	0	0	0	1
Fetch	XXXX	0	0	x	x	1	0	x	0	0	1	1	0	1	A+1	1	1

MU0 Kontroll-Logik (Steuerwerk)

Ab Zeile 3: Steuersignale für 2. Takt (Execute):

Inputs						Outputs										
Instruction	Opcode	/Reset	Step	ACCz	ACC15	Step	Asel	Bsel	ACCoe	ACCie	PCoe	Pcie	Irie	ALUfs	MEMrq	R/nW
Reset	xxxx	0	x	x	x	0	x	x	0	0	0	1	0	0	0	1
Fetch	xxxx	1	0	x	x	1	0	x	0	0	1	1	1	A+1	1	1
LDA S	0000	1	1	x	x	0	1	1	0	1	0	0	0	B	1	1
STO S	0001	1	1	x	x	0	1	x	1	0	0	0	0	x	1	0
ADD S	0010	1	1	x	x	0	1	1	1	1	0	0	0	A+B	1	1
SUB S	0011	1	1	x	x	0	1	1	1	1	0	0	0	A-B	1	1
JMP S	0100	1	1	x	x	0	x	0	0	0	0	1	0	B	0	1
JGE S	0101	1	1	x	1	0	x	x	0	0	0	0	0	x	0	1
		1	1	x	0	0	x	0	0	0	0	1	0	B	0	1
JNE S	0110	1	1	1	x	0	x	x	0	0	0	0	0	x	0	1
		1	1	0	x	0	x	0	0	0	0	1	0	B	0	1
STOP	0111	1	1	x	x	1	x	x	0	0	0	0	0	x	0	1

Ein erstes MU0 Programm

Was steht am Ende im ACC?
Was steht am Ende in Total?

```
0x0   Loop       LDA     Total       ; Accumulate total = 0
0x2   Add_instr  ADD     Table       ; Begin at head of table, ACC=39
0x4              STO     Total       ; Total (Summe)=ACC (39)
0x6              LDA     Add_instr   ; Change data address
0x8              ADD     Two         ; by modifying instruction!
0xA              STO     Add_instr   ; OpCode „ADD Table“ incremented!
0xC              LDA     Count       ; Count iterations
0xE              SUB     One         ; Count down to zero
0x10             STO     Count       ;
0x12             JGE     Loop        ; If >= 0 repeat
0x14             STP                 ; Halt execution

; Data definitions
0x16  Total      DEFW    0           ; Total - initially zero, contains sum over all table elements
0x18  One        DEFW    1           ; The number one
0x1A  Count      DEFW    4           ; Loop counter (loop 5x)
0x1C  Table      DEFW    39          ; 1st  element of table
0x1E             DEFW    25          ; 2nd  element of table
0x20             DEFW    4           ; 3rd element of table
0x22             DEFW    98          ; 4th  element of table
0x24             DEFW    17          ; 5th  element of table
0x26  Two        DEFW    2           ; the number two
```

Dieses Programm enthält sich selbst modifizierenden Code.

Tun sie das besser nicht, wenn sie nicht sehr genau wissen, was sie tun!!!

Aber: MU0 kann Arrays nur auf diese Weise verarbeiten (keine Index- oder Adresskalkulationen)

DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Die Weiterentwicklung zum MU1…