Vorlesung 2

Rechnerarchitektur

Peter B. Ladkin
ladkin@rvs.uni-bielefeld.de

Inhalt

- Allgemeine von Neumann Architektur
- Architektur der CPU
- Wie sie funktionieren
- Assembly Sprache
- Befehls-Ausführung

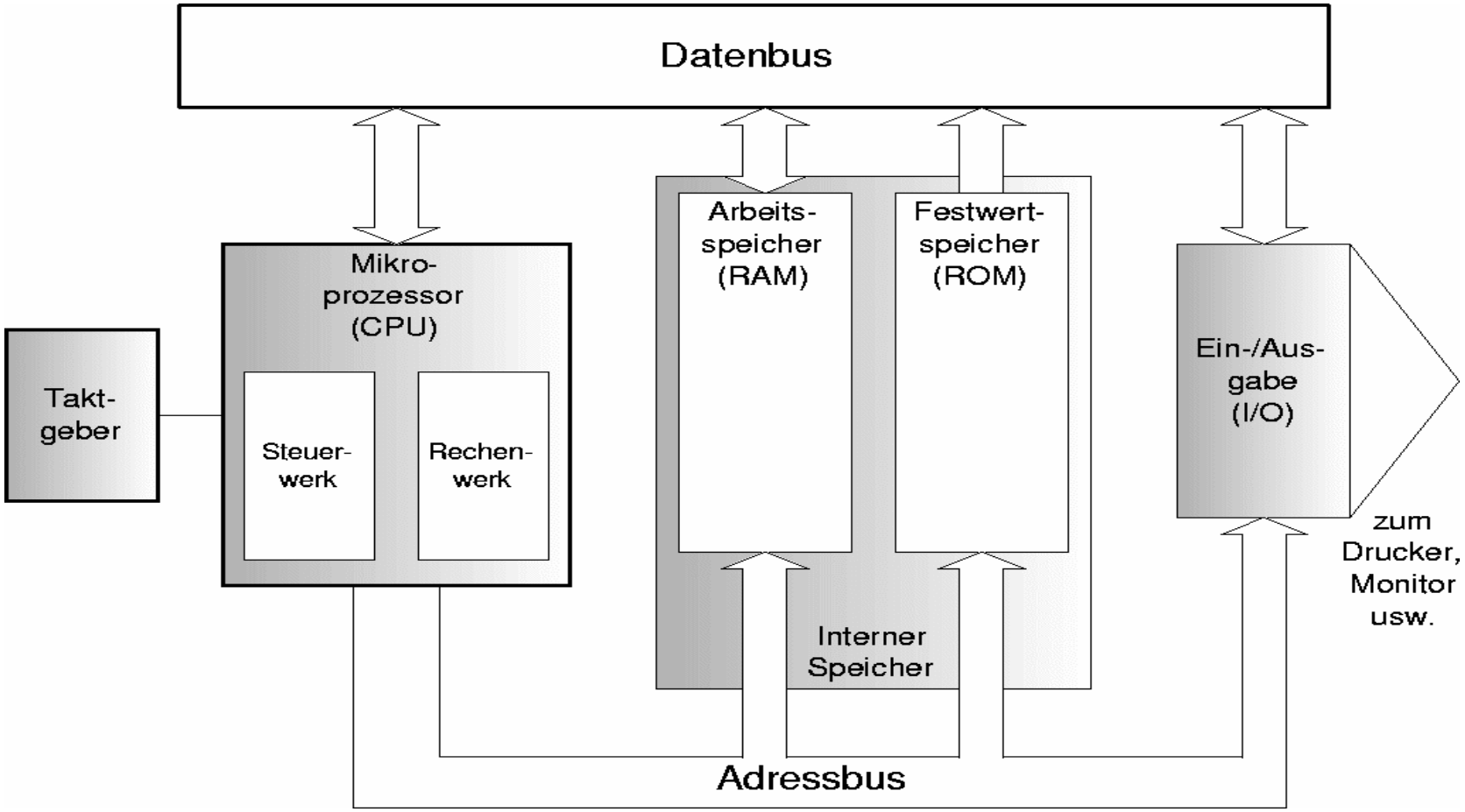
Von Neumann Architektur

- Stored Program Architektur
- Speicher beinhaltet Daten
- Speicher beinhaltet auch das Programm
- Befehle werden vom Speicher geholt
- Sowie Daten
- Speicher in Daten-/Programm-Speicher geteilt

Vorher

- Programme per Laufbandleser gelesen
- "Paper Tape" - Papierband mit Löcher
- Oder per "punched card". Hollerith (IBM)
- Auch Daten
- Aber Zwischenresultate gespeichert

Allgemeine vN-Achitektur



Adressen

- Speicher ist ein Feld von erheblicher Größe
- Adressen sind wie Feld-Indizes
- Die Speicherhardware "interpretiert" eine Adresse
- Adressen werden zum Speicher auf dem Adressbus übertragen

Daten

- Daten sind der Inhalt eines Array-Elementes
- Wenn eine Adresse "hereinkommt", wird das Datum, das in der Adresse gespeichert ist, auf den Datenbus gegeben

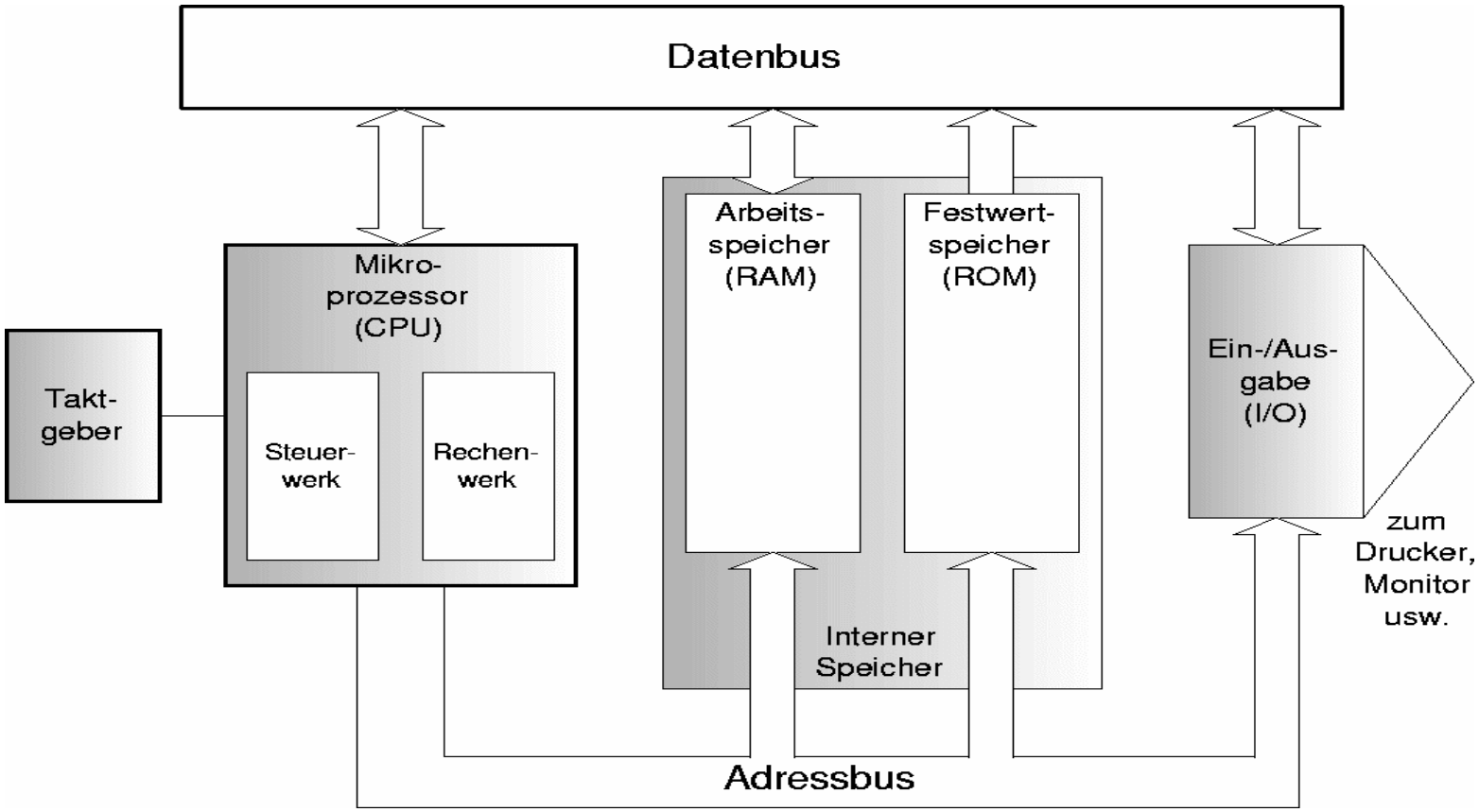
Bits und Strom

- Wie "laufen" Daten und Adressen auf dem Bus?
- Ein Beispiel:
  Daten und Adressen sind "8-bit" breit
- Ein Bus besteht auf 8 parallelen Leitungen
- Für eine "1" wird die entsprechende Leitung "hoch" gesetzt (die Spannung ist “high”)
- Für eine "0" auf null gesetzt (Spannung “low”)
- Für eine entsprechende Zeit

Bits und Strom

- Nicht die einzige Methode, aber am weitesten verbreitet
- "Zeit" kommt von einer Uhr
- Z.B. die selbe Uhr, die die Taktfrequenz gibt

Architektur nochmal



Wie es funktioniert

- Tick: Adresse auf Adress-Bus (entsprechende Leitungen hochgesetzt)
- Tick: erreicht Speicher
- Tick, Tick: Speicherelektronik öffnet Adresse
- Tick: Daten auf Datenbus eingeschaltet
- Tick: Datenwelle erreicht CPU
- Tick: CPU schaltet Daten auf ein Register ein

Wie es funktioniert

- Daten laufen auf dem Bus nicht unbedingt im Takt
- Speicher reagiert nicht undedingt auf den Takt
- Aber es gibt eine Anzahl von Ticks in dem man auf jeden Fall die Daten bekommt
- Dieser wird "Memory latency" genannt und für alle Speicher-Datenanschlussoperationen benutzt

Wie es funktioniert

- Die meisten CPUs haben internen Speicher: "Register" (wenig) und “Cache” (mehr)
- Register sowie Cache reagieren auf Taktfrequenz
- Dann muss Speicher nicht mehr synchron mit der CPU arbeiten
- Er könnte asynchron mit der CPU arbeiten

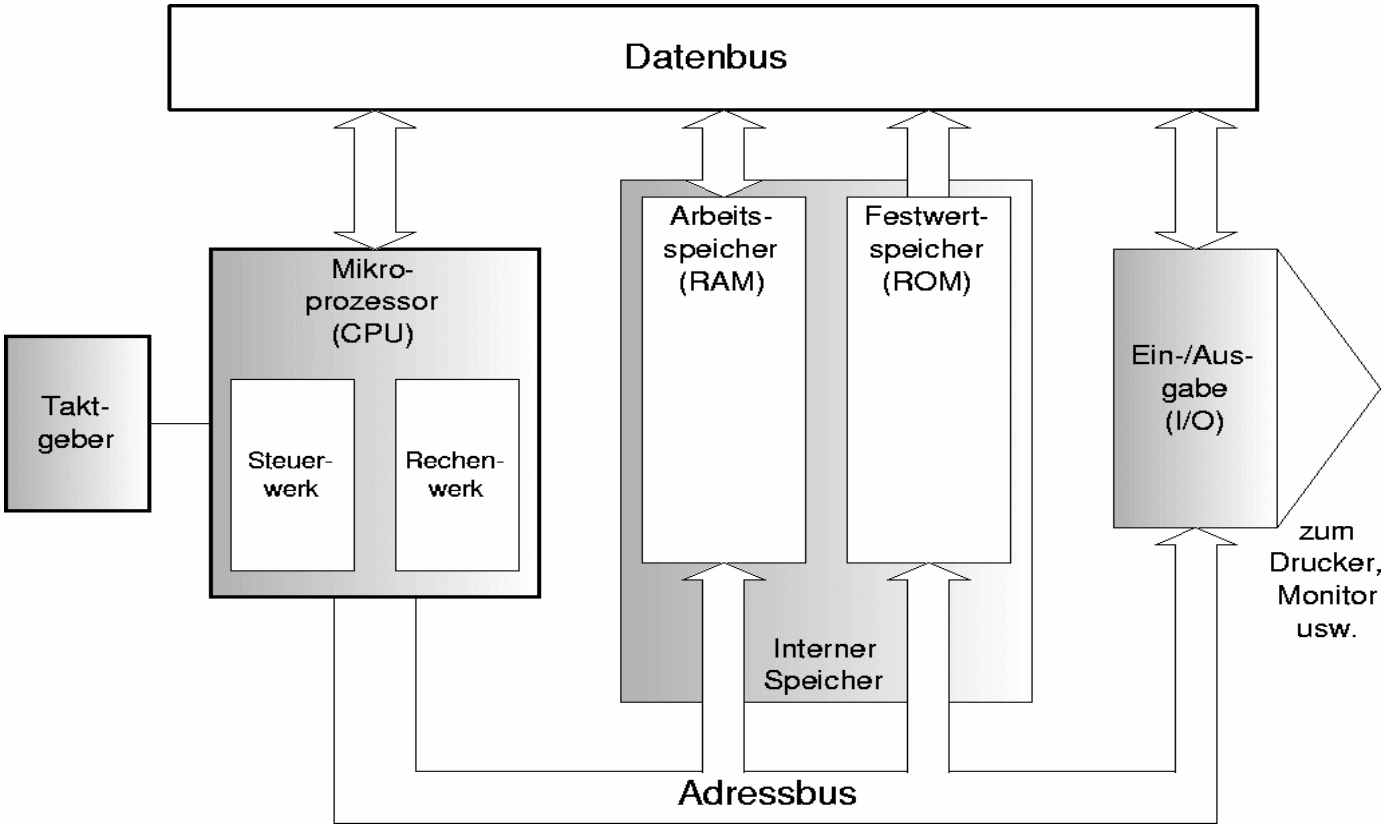
Wie es funktioniert

- Es gibt eine Menge kluger Algorithmen, die Cache und Speicher effizient benutzten
- Die Algorithmen werden in Hochleistungsprozessoren verwendet

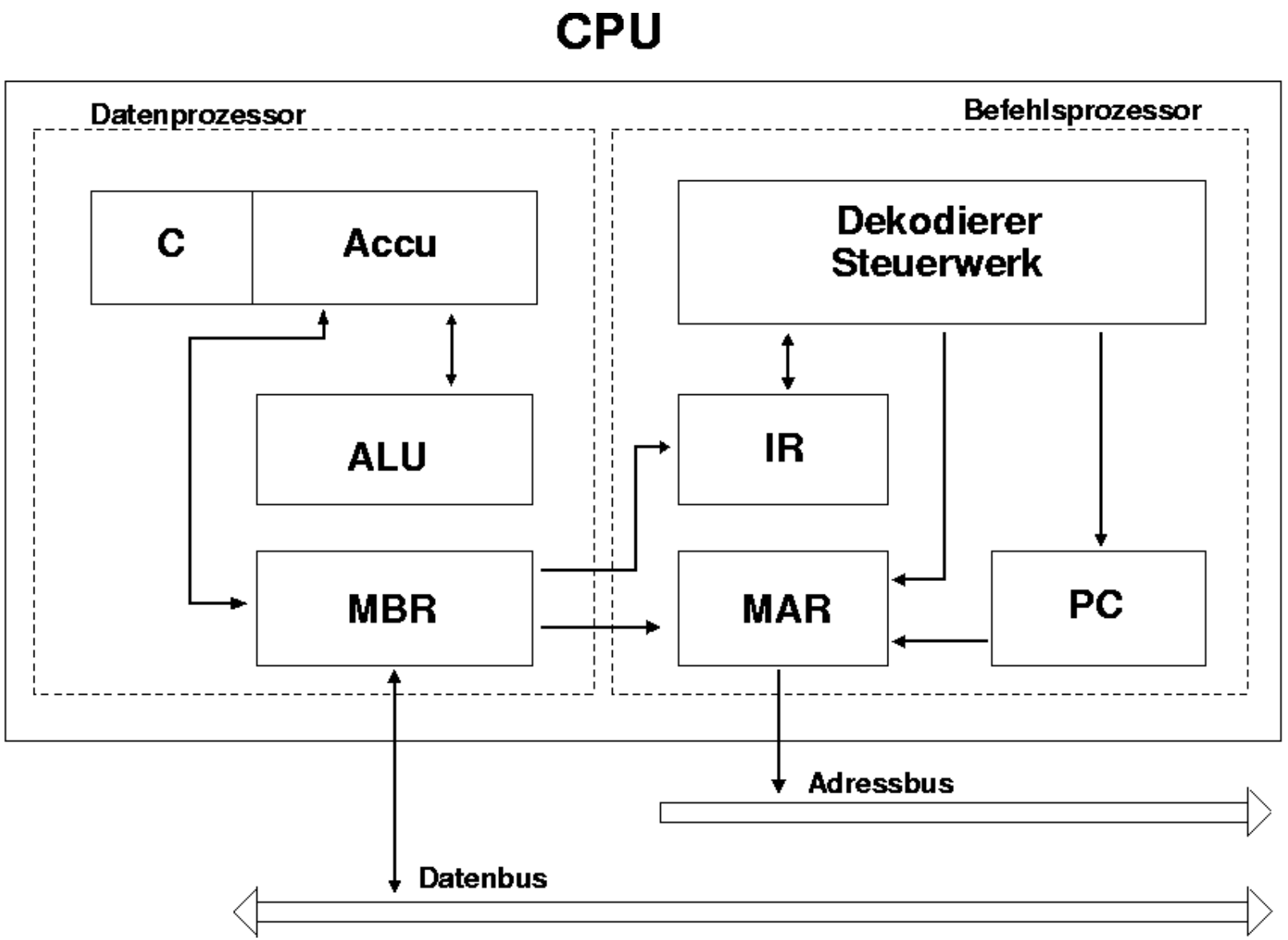
I/O

- Funktioniert wirklich asynchron mit der CPU
- Und sehr langsam
- Von Software bedient: "Drivers"
- Adressen von I/O-Geräten alle höher als Adressen vom Speicher

CPU


Datenbus
Mikro-
prozessor
(CPU)
Takt-
geber
Steuer-
werk
Rechen-
werk
Arbeits-
speicher
(RAM)
Festwert-
speicher
(ROM)
Interner
Speicher
Ein-/Aus-
gabe
(I/O)
zum
Drucker,
Monitor
usw.
Adressbus

CPU


CPU
Datenprozessor
Befehlsprozessor
C
Accu
ALU
MBR
Dekodierer
Steuerwerk
IR
MAR
PC
Adressbus
Datenbus

CPU

- PC hält die Adresse des nächsten Befehls
- Tick: wird in MAR gespeichert
- Tick: wird auf den Adressbus gestellt; PC gleichzeitig inkrementiert
- Tick...tick: Daten (Befehl) kommt in MBR rein
- Tick: Befehl wird in IR transferiert
- Tick: Befehl wird in Dekodierer transferiert
- Wird in Befehl und Argumente geteilt

CPU: "Jump"-Befehl

- Falls "Jump <Befehl-Adresse>"
- Wird in "Jump" und "<Befehl-Adresse>" geteilt
- <Befehl-Adresse> wird in MAR gesetzt
- Tick...tick neuer Befehl kommt in MBR rein
- Wird in IR transferiert
- Zurück zum Anfang

CPU: "Add"-Befehl

- Falls "Add"-Befehl
- "Add" wird an ALU weitergegeben
- "Add <Speicher-Adresse>" bedeutet:
  - ACC <- <Sp.Ad.-Inhalt> + ACC
- Speicher-Adressen werden MAR -> MBR Zyklus folgen

CPU: "Add"-Befehl

- Tick...tick; Inhalt der <Sp.-Adresse> wird in MBR erscheinen
- ALU transferiert MBR-Inhalt in die ALU
- Dieser Inhalt wird zum Inhalt des ACCs addiert innerhalb der ALU
- Resultat wird in den ACC transferiert
- PC wird in MAR transferiert usw.

Assembly-Sprache

- Es wird "Jump xxyyzz" geschrieben statt Binärcode
- Ein "Assembler" convertiert die Programme in binäre Notation
- Die binäre Notation ist die Maschinen-Sprache
- Maschinen-Sprache wird von der Maschine ausgeführt

Assembly-Sprache

- Arithmetische Operationen
  - Add: ACC <- ACC + <Sp.Ad.-Inhalt>
  - Subtract: ACC <- ACC - <Sp.Ad.-Inhalt>
  - Shift (multiply by 2): ACC <- ACC * 2 == alle Bits in ACC einmal nach links gestellt mit 0 in Low-Order Bit
  - Multiply: ACC <- ACC * <Sp.Ad.-Inhalt>
  - Divide: ACC <- ACC / <Sp.Ad.-Inhalt>

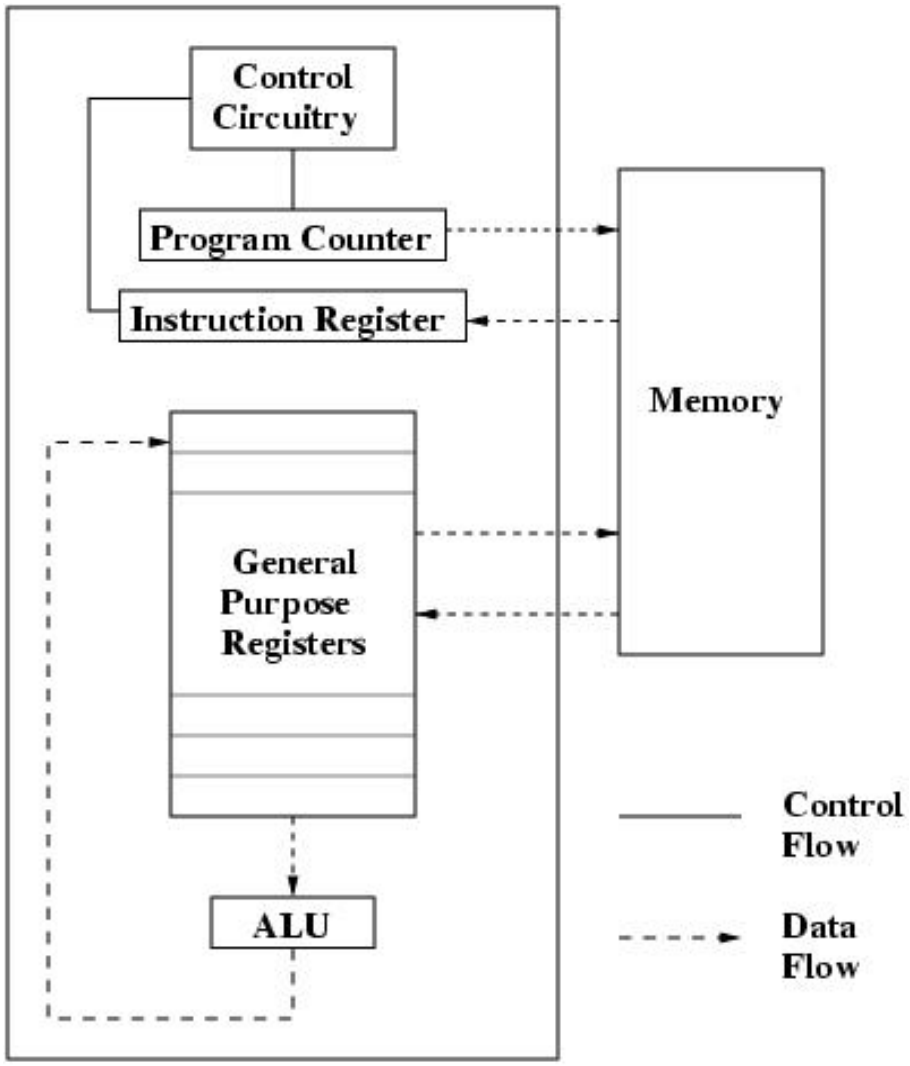
Assembly-Sprache

- Logische Operationen
  - "Jump <Befehl-Ad.>" : Inhalt der <Bef.-Ad.> wird in IR geladen; PC wird <Bef.-Ad.> + 1
  - "Load <Sp.-Ad.>": Inhalt der <Sp.-Ad.> wird in ACC geladen; PC wird PC + 1
  - "Store <Sp.-Ad.>": Inhalt des ACCs wird in <Sp.Ad.> gespeichert; PC wird PC + 1
  - "If ACC then <Bef.-Ad.>": ACC > 0 then PC wird <Bef.-Ad.>; ACC </= 0 then No-Op.

Übung

- Nimm die 9 Befehle
- Beschreibe genau, was bei jedem Tick passieren muss, um den Befehl endgültig auszuführen
- Zähl die Ticks pro Befehl
- Bau zusätzliche "null"-Ops ein, um die Anzahl der Ticks für jeden Befehl auszugleichen

Eine abstrakte CPU

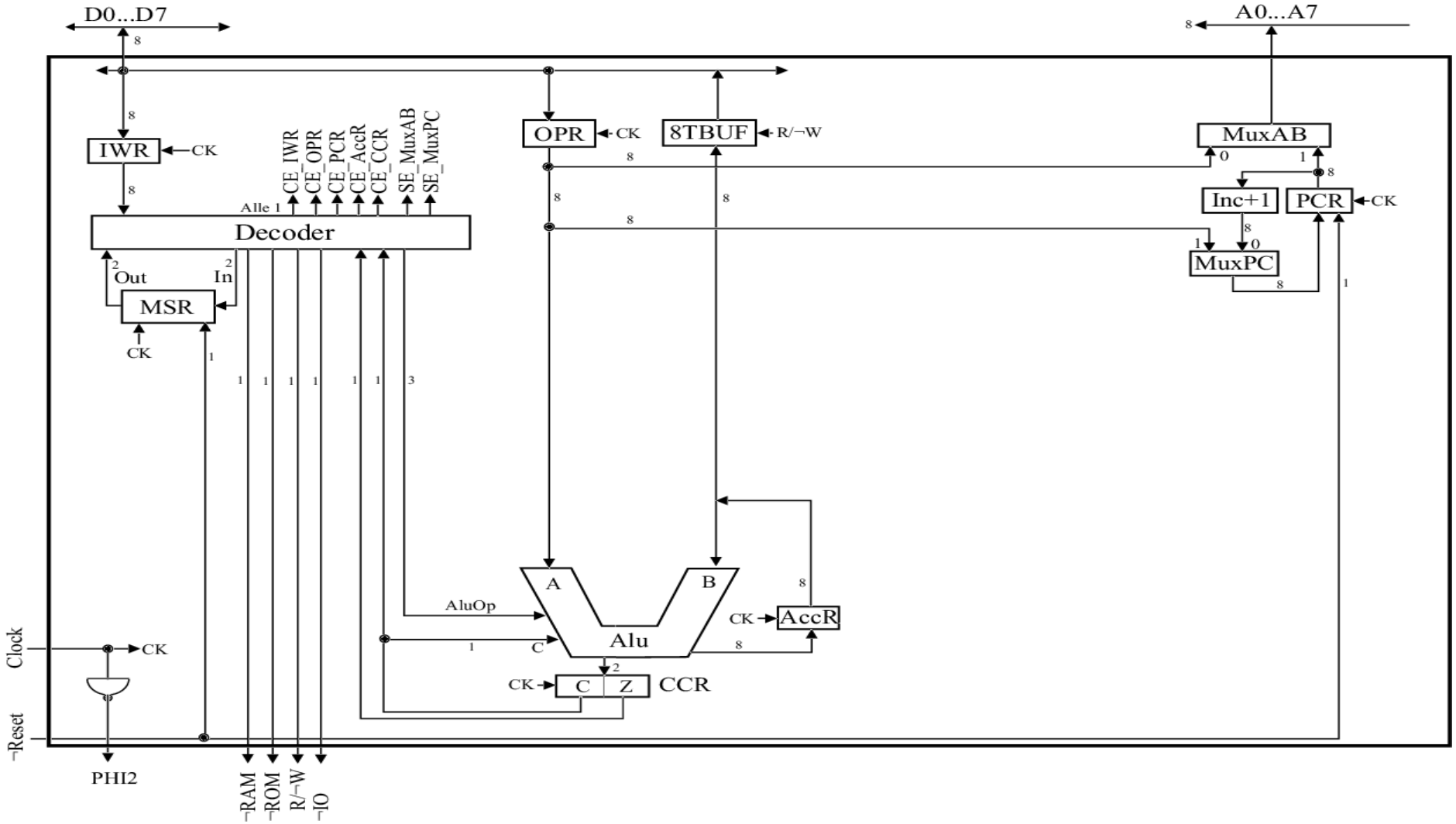


CPU

CPU - fehlende Teile

- Interrupt-Werk ist nicht beschrieben
- Genaue Funktionsweise der Uhr ist nicht beschrieben

CPU des DEPs


D0...D7
A0...A7
IWR
CK
OPR
8TBUF
R/¬W
MuxAB
Inc+1
PCR
MuxPC
CE_IWR
CE_OPR
CE_PCR
CE_AccR
CE_CCR
SE_MuxAB
SE_MuxPC
Alle 1
Decoder
Out
In
MSR
A
B
Alu
AluOp
C
AccR
CCR
Z
Clock
¬Reset
PHI2
¬RAM
¬ROM
R/¬W
¬IO