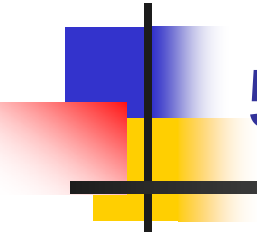


5. Der Prozessor DINATOS

Technische Grundlagen der Informatik 2
(Rechnertechnologie 2)
SS 2006

Andreas Koch

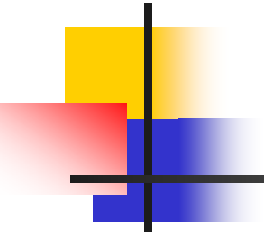
FG Eingebettete Systeme und ihre Anwendungen

Auf Basis von Material von

Rolf Hoffmann

FG Rechnerarchitektur

Technische Universität Darmstadt



Inhalt

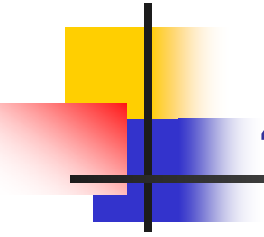
5.1 Zum Begriff Architektur

- **ISA : Instruction Set Architecture, Maschinenarchitektur, Programmiermodell**
  - Die ISA ist definiert durch
    - die Maschinenbefehle
    - deren Wirkung auf den Objekten, die der (Maschinen-)Programmierer sieht
    - insbesondere Speicher, Register und Ein-/Ausgabe-Schnittstelle.
  - Der Maschinenprogrammierer und Systemprogrammierer benutzt die ISA.
  - Die Benutzung der ISA wird in den Grundlagen der Informatik 3 gelehrt.

Unterschied zwischen Architektur und Implementierung!

* Implementierung realisiert Architektur in Hardware

* Kann auf unterschiedlichste Weisen geschehen



Zum Begriff Architektur

Mikroarchitektur, *"Innenarchitektur"*

- Die Implementierung der ISA durch Hardware-Objekte
  - Register, Speicher, Verdrahtung, Logik
- Detaillierte Beschreibung, wie Maschinenbefehle durch Hardware interpretiert werden.
- Beschleunigungsmaßnahmen durch Ausnutzung der inhärenten ILP
  - instruction level parallelism,
  - insbesondere durch Pipelining
- Beschreibungsmittel:
  - Synchrone Automaten
  - Mikrooperationen
  - Register-Transfer-Ebene
  - Verilog
- Lehrstoff dieser Vorlesung

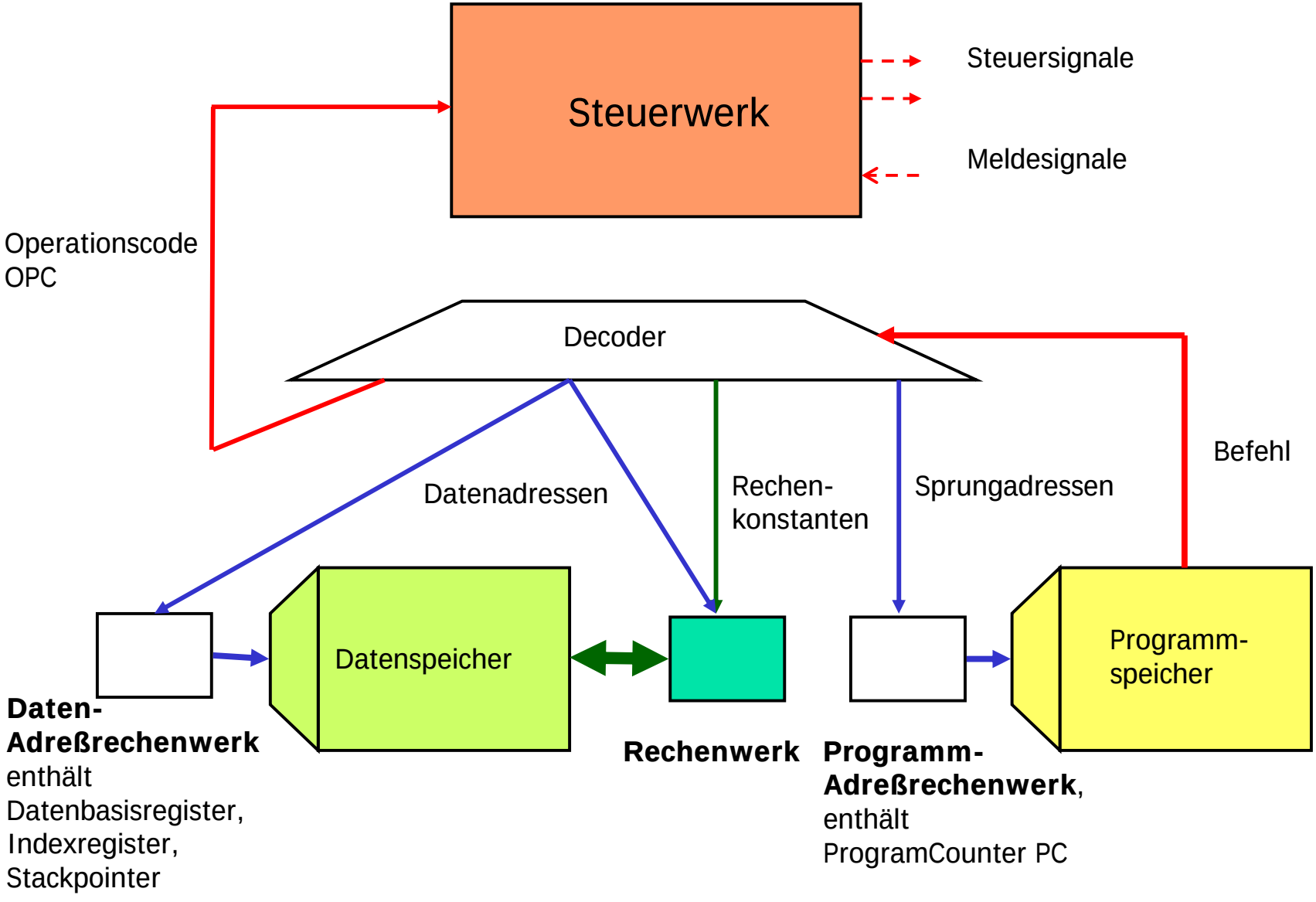
5.2 Komponenten eines Rechners

- **Peripherie**
  - Tastatur und Maus
  - Monitor
  - Drucker und Scanner
  - CD-Laufwerk
  - Netzanschluß
  - weitere wie: Bandspeicher, Kamera, Multimedia-Unterstützung, Sprachein-/ausgabe, Sensoren, Aktoren
- **Plattenspeicher**
  - zur permanenten Speicherung großer Datenmengen
  - schneller Austausch von Daten mit dem Hauptspeicher
- **Prozessor**
  - Hauptspeicher
  - CPU: Central Processing Unit, Zentraleinheit

5.3 Speicher und Prozessor

- **Hauptspeicher**
  - Datenspeicher
    - speichert die Daten in binärer Form
  - Programmspeicher
    - enthält das Maschinenprogramm (Objektcode) in binärer Form
    - besteht aus einzelnen Maschinenbefehlen
  - (Harvard-Rechner: Datenspeicher u. Programmspeicher getrennt)
  - (von Neumann-Rechner: Datenspeicher u. Programmspeicher vereinigt)
- **CPU**
  - Kontrolleinheit (Steuerwerk, Leitwerk)
    - interpretiert die Maschinenbefehle
      - Hole den nächsten Befehl
      - Decodieren (Zerlegen in seine Bestandteile)
      - Ausführen durch eine Folge von Mikrooperationen
  - Rechenwerk
    - Operationen auf den Daten

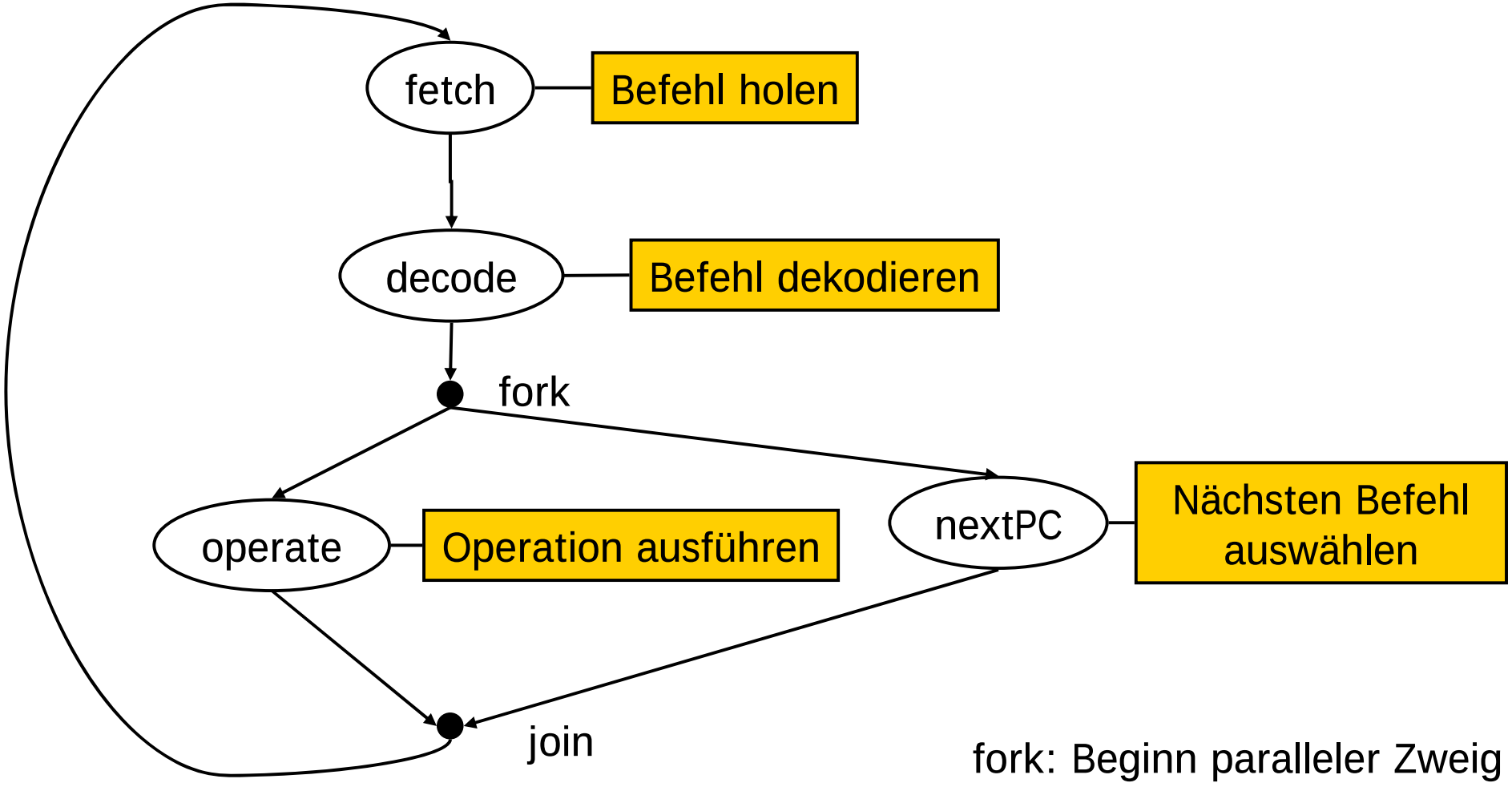
5.3 Prozessor



Erklärung zum Prozessor

- Das **Steuerwerk** hat die zentrale Kontrolle
  - veranlaßt das Holen des nächsten Befehls
    - Programmspeicher wird mit dem Programmzähler adressiert
  - veranlaßt das Decodieren
    - OPC: Operationscode
    - Datenadressen: dienen zur Adressierung der Daten im Datenspeicher
    - Rechenkonstanten: werden an das Rechenwerk weitergeleitet
    - Sprungadressen: dienen zur Verzweigung im Programm
  - veranlaßt die Ausführung
    - Transport von Daten zwischen Datenspeicher und Rechenwerk
    - Operation im Rechenwerk
- **Daten-Adreßrechenwerk**
  - benutzt Datenadressen aus dem decodierten Befehl
  - berechnet die Datenadresse
  - enthält mindestens ein Register zur Adressierung des Datenspeichers
  - enthält weitere Register wie Stackpointer
- **Programm-Adreßrechenwerk**
  - benutzt Sprungadressen aus dem decodierten Befehl
  - berechnet die nächste Programmadresse (wo der nächste Befehl zu holen ist)
  - enthält mindestens den Programmzähler
- **Rechenwerk**
  - enthält Datenregister (z.B. Akkumulator)
  - führt die Rechenoperationen auf den Registern (adressiert durch Registeradressen) aus
  - tauscht Daten mit dem Hauptspeicher aus

5.4 Interpretationszyklus



fork: Beginn paralleler Zweig

join: Ende paralleler Zweig

Die Operationen im Interpretationszyklus

- Befehl holen
  - Programmspeicher **pm** wird durch Befehlszähler (program counter) **pc** adressiert
  - Befehl an Adresse **pc** wird geholt (memory read)
  - In Befehlsregister zwischengespeichert. **bef<=pm[pc]**
- Befehl dekodieren: bef wird in Bestandteile zerlegt.
  - **OPC** = Operation Code, die auszuführende Operation
  - **Datenadressen** für Operanden, falls sie aus dem Datenspeicher in das Rechenwerk geholt werden oder Ergebnisse vom Rechenwerk in den Datenspeicher geschrieben werden.
  - **Rechenkonstanten**, die als sogenannte Direktoperanden direkt in der Rechenoperation benutzt werden.
  - **Registeradressen** (Datenadressen) zur Auswahl der Register im Rechenwerk.
  - **Sprungadressen** (**label)**, die dazu benuzt werden, den Befehlszähler auf ein Sprungziel zu setzen

- Operation ausführen
  - Die eigentliche Operation wird ausgeführt,
  - Operationen auf Daten im Rechenwerk
  - Austausch von Daten mit dem Datenspeicher (load, store).

- Nächsten Befehl auswählen
  - Normalerweise wird der Befehlszähler um eins erhöht (**pc<=pc+1**).
  - Dadurch wird der nächste Befehl ausgewählt.
  - Im Falle eines Sprungbefehls (unbedingt oder bedingt) wird der Befehlszähler auf das Sprungziel gesetzt (**pc<=label**)

5.5 DINATOS

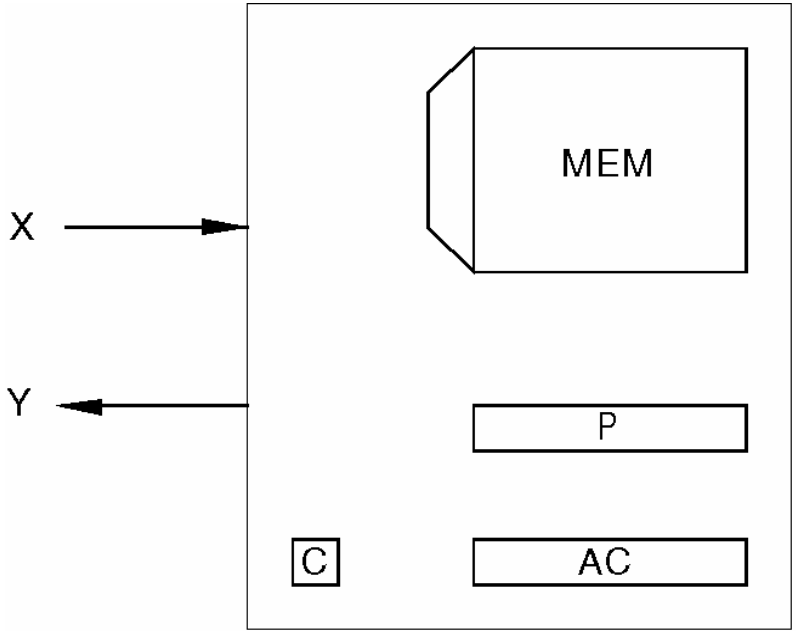
- DINATOS ist ein einfacher Prozessor. Seine Arbeitsweise wird in Form eines Zustandsdiagramms und des entsprechenden Verilog-Programms erklärt. Wir werden dadurch gut verstehen, wie ein Rechner intern die Befehle abarbeitet. Danach sind wir bestens gerüstet, um die folgenden Kapitel (MIPS-Rechner) gut verstehen zu können.

5.5.1 Hardware-Objekte

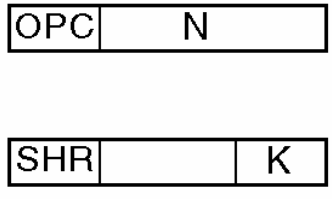
- ISA-Objekte, (Objektmenge) allgemein
  - Die *"logischen Objekte"* , auf die die Maschinenbefehle zugreifen
  - Speicher, Register, ALU, Funktionseinheiten, Datenträger, EA-Schnittstelle

- Objektmenge des Beispiel-Rechners DINATOS

MEM	Hauptspeicher
P	Befehlszähler
AC	Akkumulator
C	Condition-Bit
X	Eingangssignale
Y	Ausgangssignale



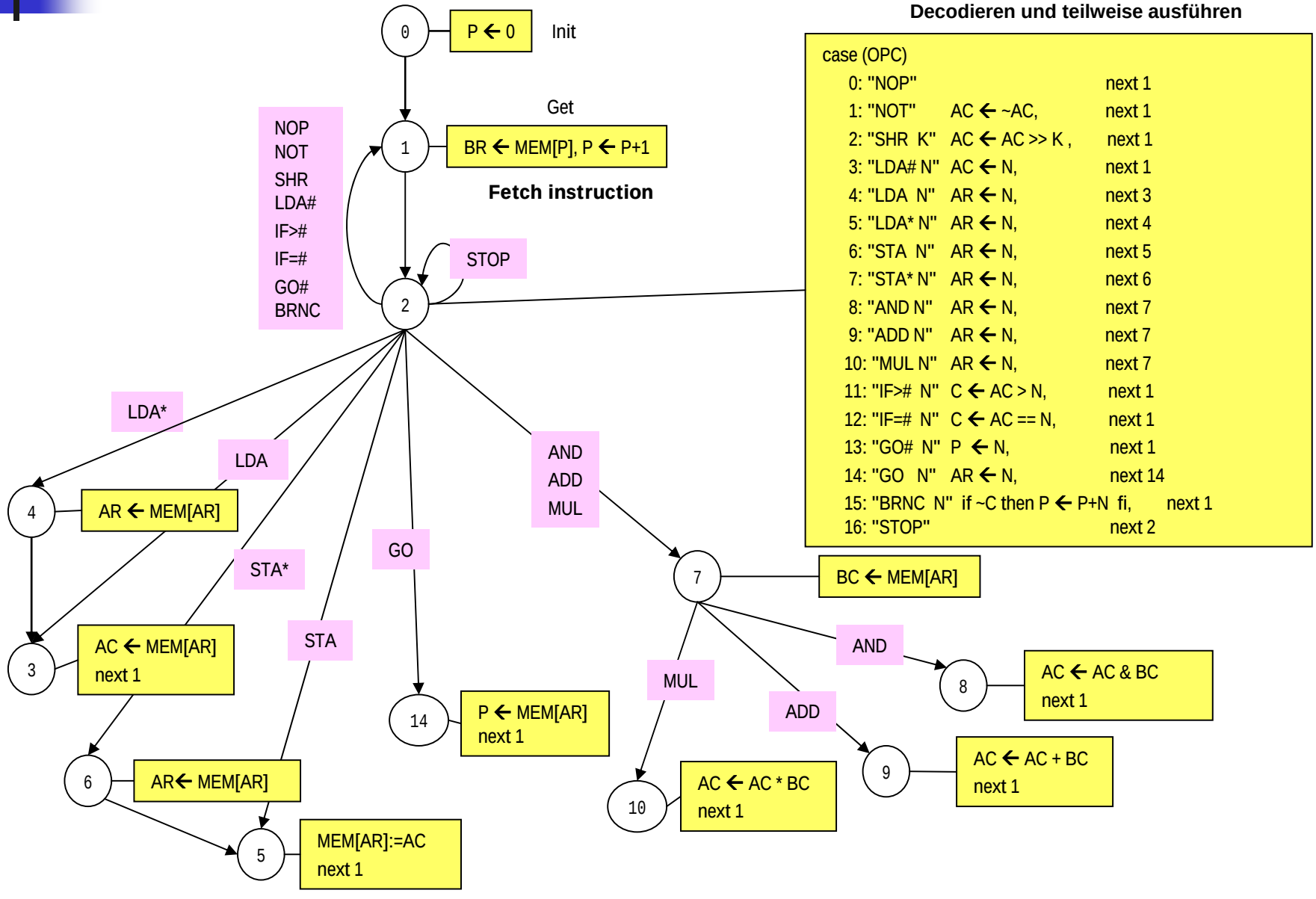
Befehlsstruktur



5.5.2 Befehlssatz

OPC (hex)	OPC		Wirkung	Beschreibung
0	NOP			No Operation
1	NOT		AC ← ~AC	Not AC
2	SHR	K (Shiftdistanz)	AC ← AC>>K	Shift Right AC
3	LDA#	N (Konstante)	AC ← N	Load AC Const
4	LDA	N (Adresse)	AC ← mem[N]	Load AC MemOperand
5	LDA*	N (Adresse)	AC ← mem[mem[N]]	Load AC indirect
6	STA	N (Adresse)	mem[N] ← AC	Store AC
7	STA*	N (Adresse)	mem[mem[N]] ← AC	Store AC indirect
8	AND	N (Adresse)	AC ← AC & mem[N]	AC and MemOperand
9	ADD	N (Adresse)	AC ← AC + mem[N]	AC add MemOperand
A	MUL	N (Adresse)	AC ← AC * mem[N]	AC mul MemOperand
B	IF>#	N (Konstante)	C ← AC > N	Test if greater Const
C	IF=#	N (Konstante)	C ← AC == N	Test if equal Const
D	GO#	N (Sprungziel)	P ← N	Goto N
E	GO	N (Adresse)	P ← mem[N]	Goto indirect
F	BRNC#	N (Konstante)	if (~C) P←P+N+1 else P←P+1	Branch if not Condition
10	STOP			Stop

5.5.3 Zustandsdiagramm



5.5.4 Mikrooperationen

AC ← ~AC, AC >>K, N, MEM[AR], AC&BC, AC+BC, AC*BC

BC ← MEM[AR]

C ← (AC>BC), (AC==BC)

BR ← MEM[P]

P ← 0, P+1, N, P+N, MEM[AR]

AR ← N, MEM[AR]

MEM[AR]:= AC (asynchron)

5.5.5 Operationswerk

- ALU-Funktionen
  - NOT, >>K , UND, +, *
- Vergleichsfunktionen
  - >, =
- Adreßrechenwerk
  - P+1, P+N, 0
- Speicher
  - Speicheradressierung mit AR, P
  - Lesen des Speicherinhalts nach AC, BC, AR, BR
  - Schreiben von AC
- Verbindungen

5.5.6 Beispiel Maschinenprogramm

Adresse	Inhalt hex	OPC	Adresse Konstante		Operation	Ergebnis
						initial AC = 0
0	00 000000	nop		No Operation		
1	01 000000	not		Invertiere AC	AC ← ~0	AC = ffffffff
2	05 000008	lda*	8	Lade indirekt	AC ← m[m[8]]	AC = 01010101
3	0b 000555	if>#	hex555	Test	C ← (AC>555)	C = 1
4	10 000000	stop				
8	00000009					
9	01010101					

5.5.7 Verilog-Beschreibung (1)


```
module dinatos(clock,stopped);

input clock;
output stopped;
reg stopped;

reg [31:0] mem [0:255]; // Speicher nur 256 Worte
reg [31:0] ac;          // Akku
reg [31:0] p;           // Befehlszähler
reg c;                  // Conditionflag
reg [31:0] bc;          // Hilfsakku
reg [31:0] br;          // Befehlsregister
reg [31:0] ar;          // Adressregister

reg [4:0] state;        // Zustand

wire [7:0]  opc;        // Befehlscode
wire [23:0] n;          // Adresse, Konstante
wire [4:0]  k;          // Shiftdistanz

assign opc = br[31:24]; // Befehlsfeldaufteilung
assign n   = br[23:0];
assign k   = br[4:0];
```

```
parameter
    nop   = 8'd0,
    anot   = 8'd1,  //  anot=not
    shr_k = 8'd2,
    ldan  = 8'd3,
    lda_n = 8'd4,
    ldain = 8'd5,
    sta_n = 8'd6,
    stain = 8'd7,
    and_n = 8'd8,
    add_n = 8'd9,
    mul_n = 8'd10,
    ifgtn = 8'd11,
    ifeqn = 8'd12,
    gon   = 8'd13,
    goin  = 8'd14,
    brnc  = 8'd15,
    stop  = 8'd16;

initial
begin
    stopped=1'b0; p=0; ac=0; bc=0; state=1;
end
```

Verilog-Beschreibung (2): Zustandsautomat


```
always @(posedge clock)
begin
    case(state)
    1:  begin // Befehlsfetch
         br<=mem[p]; p<=p+1; state<=2;
        end

    2:  begin // Befehlsdekodierung
        case(opc)
        nop   :  state<=1; // no Operation

        anot  : begin // Invertierung
                   ac<=~ac;
                   state<=1;
                end
        shr_k : begin // Shift um k Stellen
                   ac<=ac>>k;
                   state<=1;
                end
        ldan  : begin // Lade Konstante
                   ac<={8'b0000,n};
                   state<=1;
                end
        lda_n : begin // Lade Inhalt Speicherzelle
                   ar<={8'b0000,n};
                   state<=3;
                end
```

```
ldain : begin // Lade Inhalt vom Inhalt Speicherzelle
                ar<={8'b0000,n};
                state<=4;
            end
sta_n : begin // Schreibe Speicherzelle
                ar<={8'b0000,n};
                state<=5;
            end
stain :  begin // Schreibe Speicherzelle indirekt
                ar<={8'b0000,n};
                state<=6;
             end
and_n : begin // log. UND
                ar<={8'b0000,n};
                state<=7;
             end
add_n : begin  // Addition
                ar<={8'b0000,n};
                state<=7;
             end
mul_n : begin // Multiplikation
                ar<={8'b0000,n};
                state<=7;
              end
```

Verilog-Beschreibung (3)


```
ifgtn :    begin // Teste Akkuinhalt > Konstante
              c<= ac > {8'b0000,n};
              state<=1;
           end

ifeqn :    begin // Teste Akkuinhalt = Konstante
              c<= ac == {8'b0000,n};
              state<=1;
           end
```

```
gon   : begin // Sprung nach n
              p<={8'b0000,n};
              state<=1;
              end
goin  : begin // indirekter Sprung
              ar<={8'b0000,n};
              state<=14;
              end
brnc   : begin // bedingter rel. Sprung
              if(c) begin
                    state<=1;
                    end
              else  begin
                    p<=p+{8'b0000,n};
                    state<=1;
                    end
              end
stop  : begin //stop simulation
              stopped<=1'b1;
              end
endcase
end
```

Verilog-Beschreibung (4)


```
3:  begin
      ac<=mem[ar];  // read mem to ac
      state<=1;
    end

4:  begin
      ar<=mem[ar];  // read address for ldain
      state<=3;
    end

5:  begin
      mem[ar]=ac;   // write ac to mem asyn
      state<=1;
     end

6:  begin
      ar<=mem[ar];  // read address for stain
      state<=5;
     end

7:  begin
      bc<=mem[ar];  // read mem operand to bc (temp)
      case(opc)
        and_n: state<=8;
        add_n: state<=9;
        mul_n: state<=10;
      endcase
    end
```

```
8:  begin                          // and
      ac<=ac & bc;
      state<=1;
    end

9:  begin                          // add
      ac<=ac+bc;
      state<=1;
    end

10: begin                          // mul
      ac<=ac*bc;
      state<=1;
     end

14: begin                          // goin
       // read target address from mem to p
       p<=mem[ar];
       state<=1;
    end
endcase
end

endmodule
```