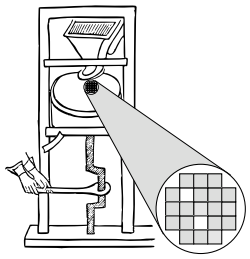
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange
Mathias Halbach (FG Rechnerarchitektur)



15.06.2006

Technische Grundlagen der Informatik II
6. Übung – MIPS und Dinatos in Verilog
Sommersemester 2006

Aufgabe 1: Dinatos-Modell

Erweitern Sie das Verilog-Modell von Dinatos (Vorlesung Kapitel 5, Folien 18 ff.) um folgende Befehle:

a) **`ADD# N`**, Addiere die Konstante N zum Akkumulator AC, AC ← AC + N.

b) **`IF= N`**, Vergleiche den Inhalt der Speicherzelle N mit dem Akkumulator AC und speichere das Ergebnis in Condition, C ← AC == mem[N].

c) **`SUBA N`**, Subtrahiere den Akkumulator vom Inhalt der Speicherzelle N, AC ← mem[N] – AC. Greifen Sie dazu auf die schon vorhandene Addition **`ADD N`** und eine zusätzliche Mikrooperation "ac ← ac + 1" zurück. Hinweis: Die ALU kann *nicht* direkt subtrahieren, es existiert keine Mikrooperation "–".

Lösung:

```
module dinatos(clock, stopped);
...
  parameter
...
  brnc   = 8'd15,
  stop   = 8'd16,  // Zeile geändert
  addn   = 8'd17,  // neu
  ifeq_n = 8'd18,  // neu
  sub_n  = 8'd19;  // neu

...
  always @(posedge clock)
  begin
```

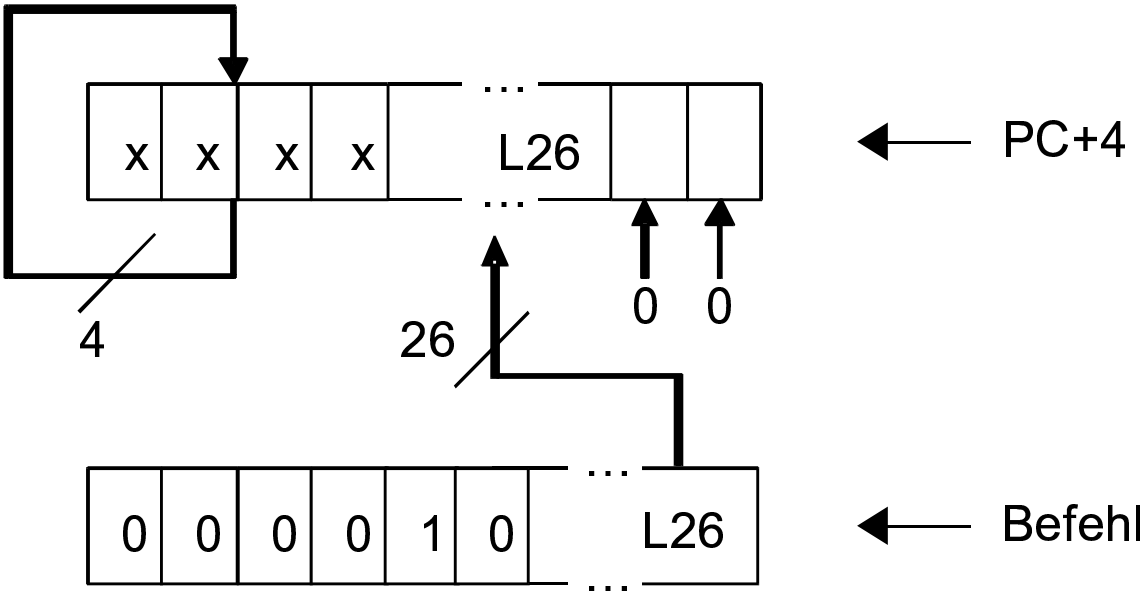
```
    case(state)
...
    2: begin // Befehlsdekodierung, ergänzt
         case(opc)
...
         addn:   begin // Neu: Addiere Konstante
                   ac    <= ac + {8'b0000,n};
                   state <= 1;
                 end
         ifeq_n: begin // Neu: Teste Akkuinhalt == Speicherzelle
                   ar    <= {8'b0000,n};
                   state <= 7;
                 end
         sub_n:  begin // Neu: Subtrahiere Speicherzelle
                   ar    <= {8'b0000,n};
                   ac    <= ~ac;
                   state <= 7;
                 end
         endcase
       end
...
    7: begin // ergänzt
         bc <= mem[ar]; // read mem operand to bc (temp)
         case(opc)
           and_n:  state <= 8;
           add_n:  state <= 9;
           mul_n:  state <= 10;
           ifeq_n: state <= 15;
           sub_n:  begin
                     ac <= ac + 1;
                     state <= 9;
                   end
         endcase
       end
...
    15:begin // neu
         c     <= ac == bc;
         state <= 1;
       end
    endcase
  end

endmodule
```

Aufgabe 2: MIPS-Sprungbefehle

In MIPS gibt es unter anderem den bedingten Sprung bei Gleichheit (**`beq`**, Opcode 4) und den direkten Sprung (Jump, **`j`**, Opcode 2). Der Wire-Bus **`OPC`** ist analog zum Verilog-Modell von Dinatos der Abgriff vom Instruktionsregister für den Opcode, **`L26`** der Abgriff für die Zielsprungadresse des Jump-Befehls und **`L16`** für den 16-Bit-Offset des Branch-Befehls. **`C`** steht in den folgenden Betrachtungen für das Ergebnis der Bedingung (Gleichheit der Register **`rs`** und **`rt`**).

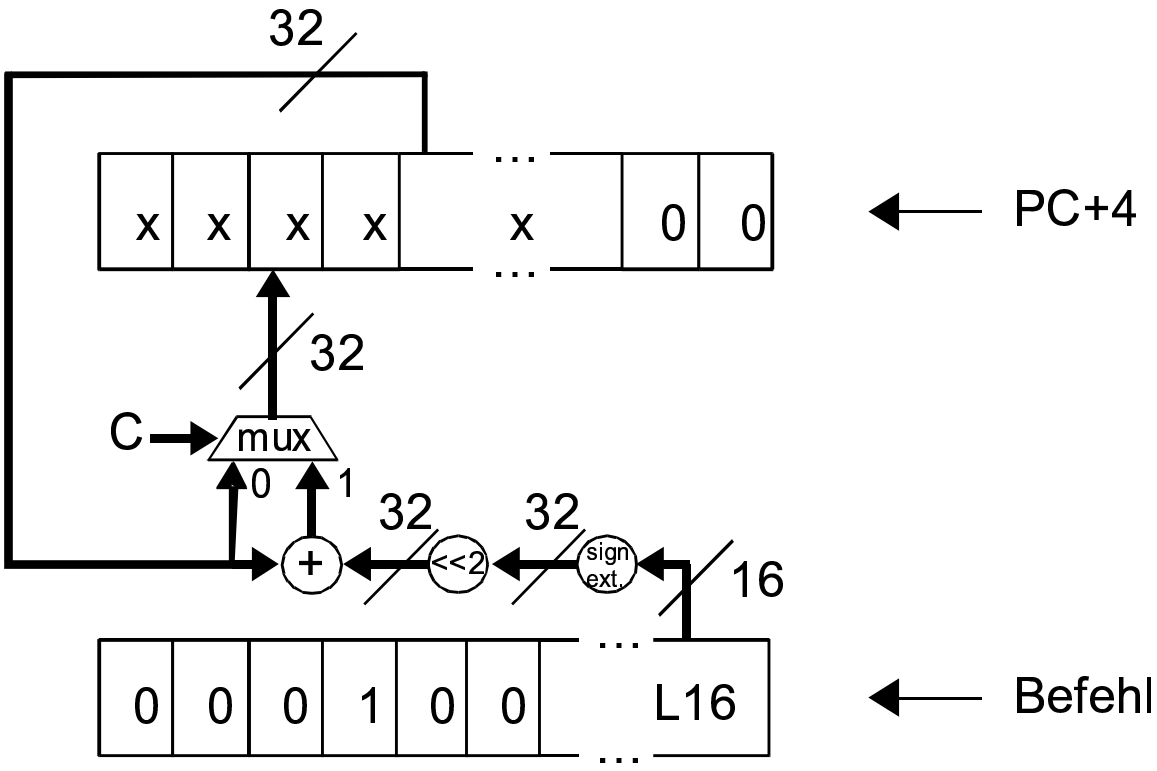
a) Zeichnen Sie das Register `PC` (das bereits den um 4 erhöhten Wert beinhalten soll), das Befehlsregister mit dem Befehl `j L26`, sowie die für nur den Jump-Befehl benötigte Logik. Die Verbindungen sollen für jedes Bit einzeln ersichtlich gezeichnet werden.

Lösung:



b) Zeichnen Sie Register und Logik analog zu a) für den `beq rs,rt,L16`-Befehl.

Lösung:



c) Formulieren Sie die auszuführenden Mikrooperationen in Verilog, die für die beiden Befehle **`jump`** und **`beq`** notwendig sind. Ihnen steht dafür eine Funktion **`ms4`** zur Verfügung, die die vier höchstwertigen Bits des übergebenen Arguments (32 Bit breit) liefert.

Lösung:

```
if (OPC == 2)
  PC <= { ms4(PC + 4), L26, 2'b00 };
else
if (OPC == 4 && C)
  PC <= { PC + 4 + {{ 14{L16[15]}}, L16, 2'b0 } };
else
  PC <= PC + 4;
```