# Übung zur Vorlesung Technische Grundlagen der Informatik

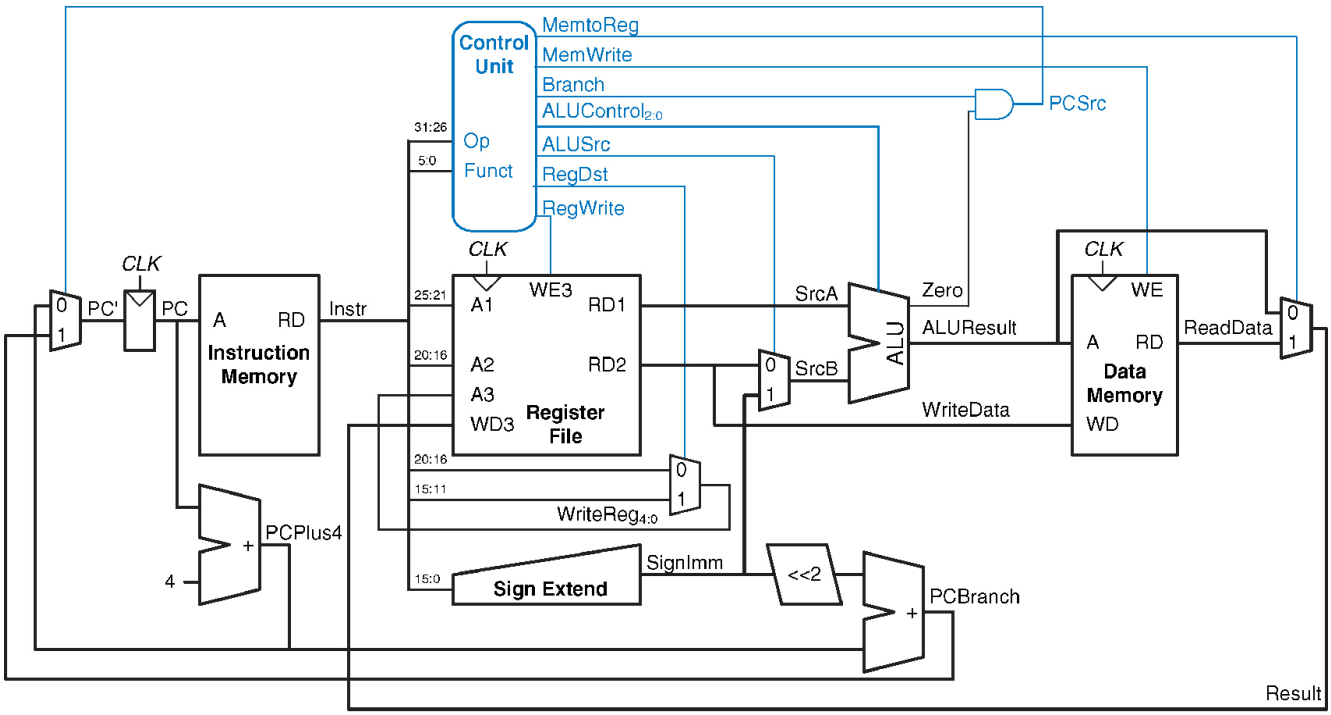
Prof. Dr. Andreas Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

**Wintersemester 09/10**
Übungsblatt 13 - Lösungsvorschlag

## Aufgabe 13.1 Erweiterung des Eintakt-MIPS-Prozessors

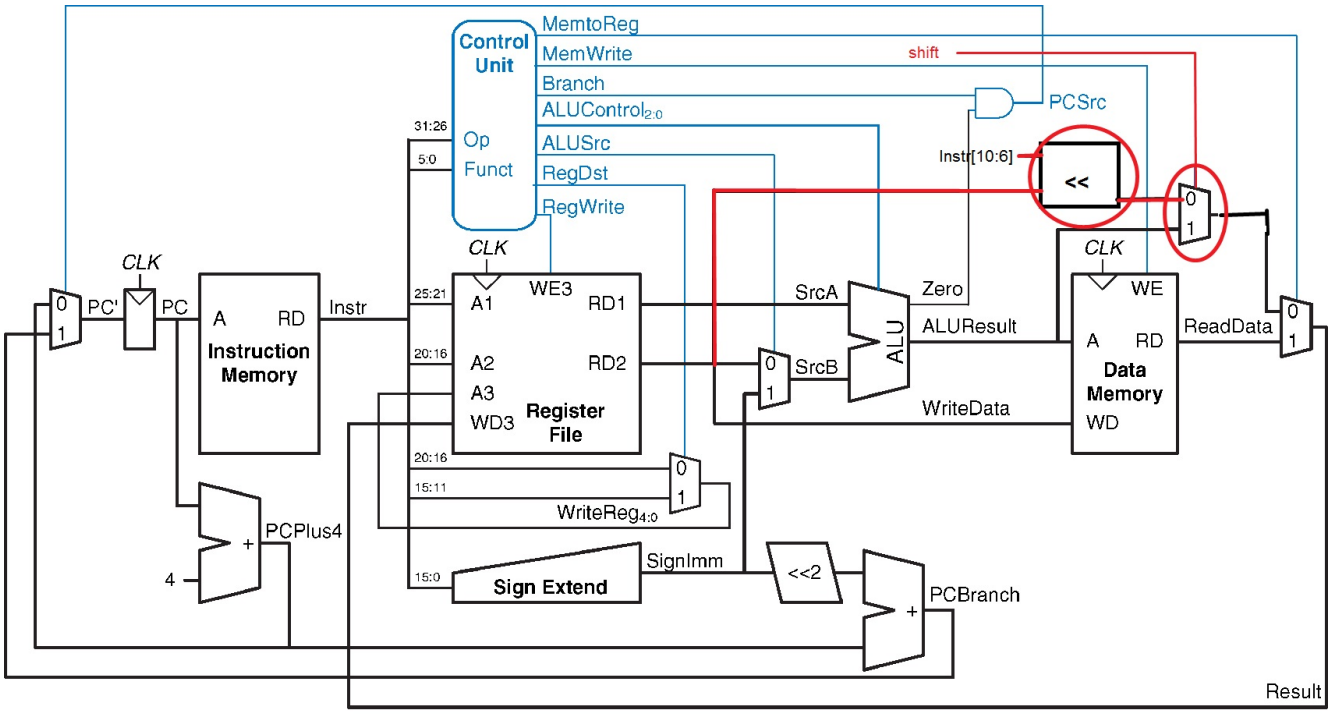
Gegeben ist die Eintakt-Implementierung des MIPS-Prozessors:






Erweitern Sie den Prozessor, so dass er den Befehl `sll` verarbeiten kann. Erweitern Sie hierzu das Schaltbild und geben Sie die Belegung der Steuersignale an.

Es muss eine zusätzliche Einheit zum Shiften hinzugefügt werden. Sie erhält als Eingang das `shamt`-Feld aus dem Befehl (`instruction[10:6]`) und den zu shiftenden Wert, der Ausgang ist der verschobene Wert. Ein neuer Multiplexer entscheidet, ob das Ergebnis aus der ALU oder dem Shifter kommt. Hierzu muss das Steuerwerk ein neues Steuersignal `shift` zur Verfügung stellen. Es ist 1, wenn das Ergebnis der ALU weitergeleitet werden soll und 0 bei Shiftoperationen (kann man natürlich auch andersherum implementieren). Die Belegung der Steuersignale: `MemtoReg` = 0, `MemWrite` = 0, `Branch` = 0, `ALUControl` = XXX, `ALUSrc` = X, `RegDst` = 1, `RegWrite` = 1.

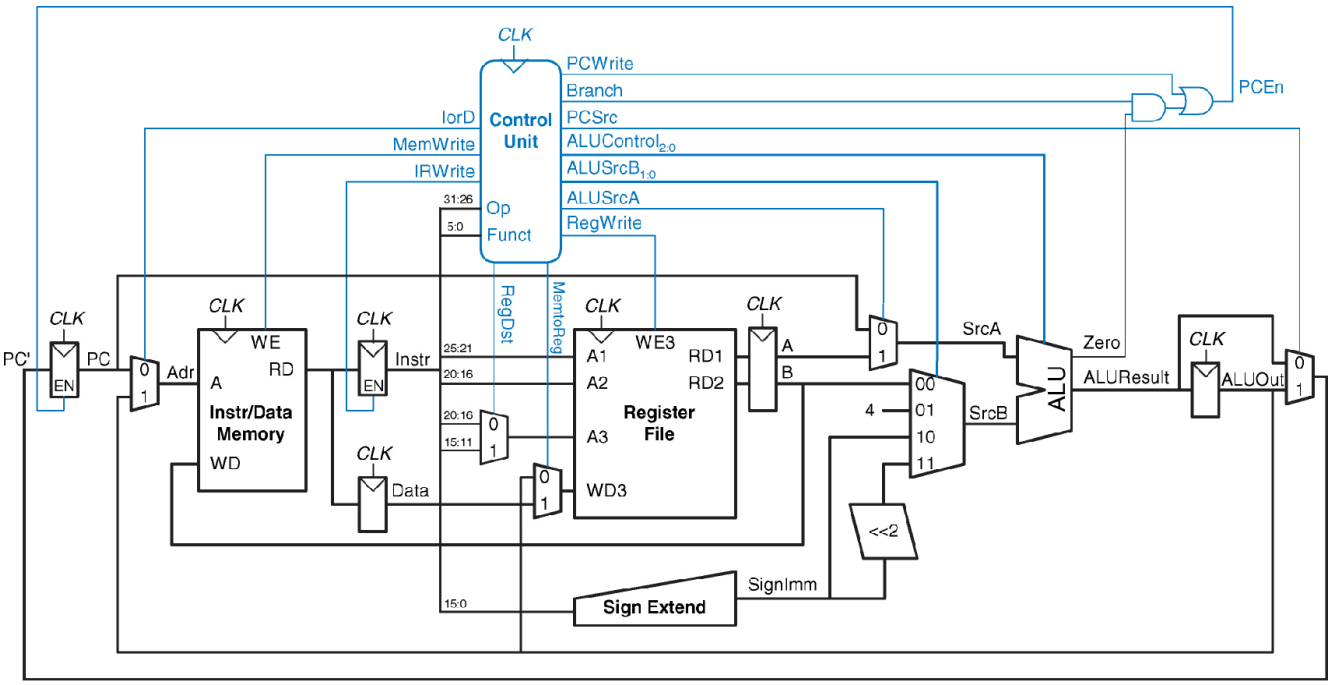




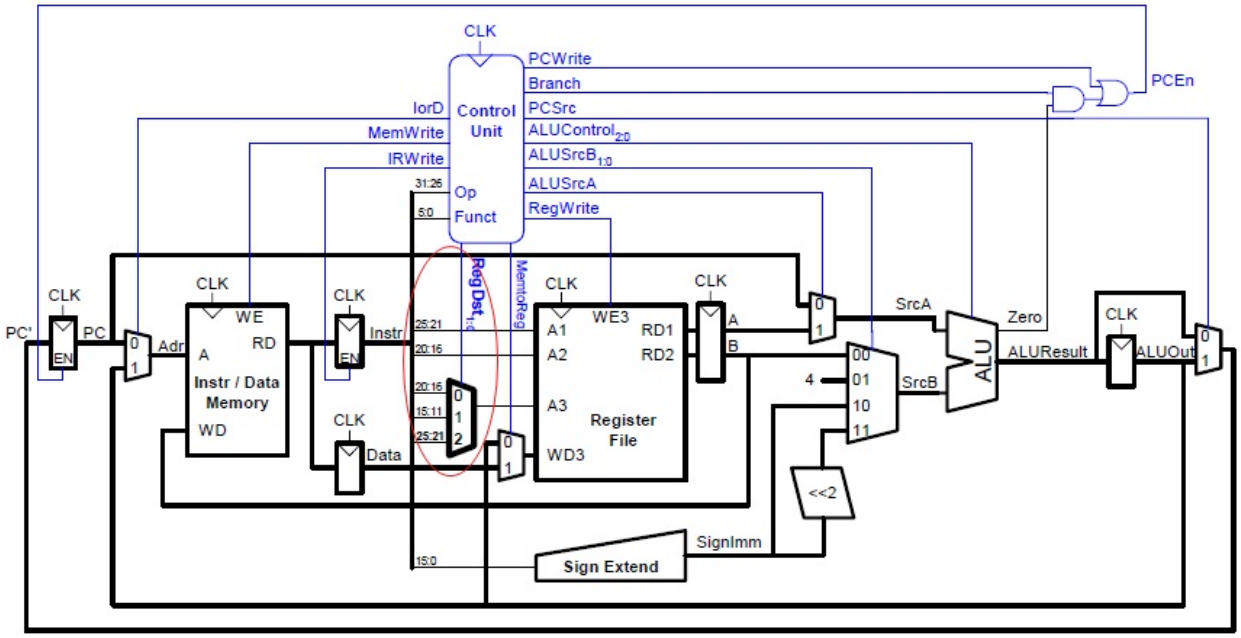

## Aufgabe 13.2 Erweiterung des Mehrtakt-MIPS-Prozessors

Erweitern Sie die Mehrtakt-Implementierung, so dass sie den Befehl `lwinc` verarbeiten kann. Dieser Befehl hat die Syntax `lwinc $rt, imm($rs)` und kombiniert die beiden folgenden Befehle zu einem:

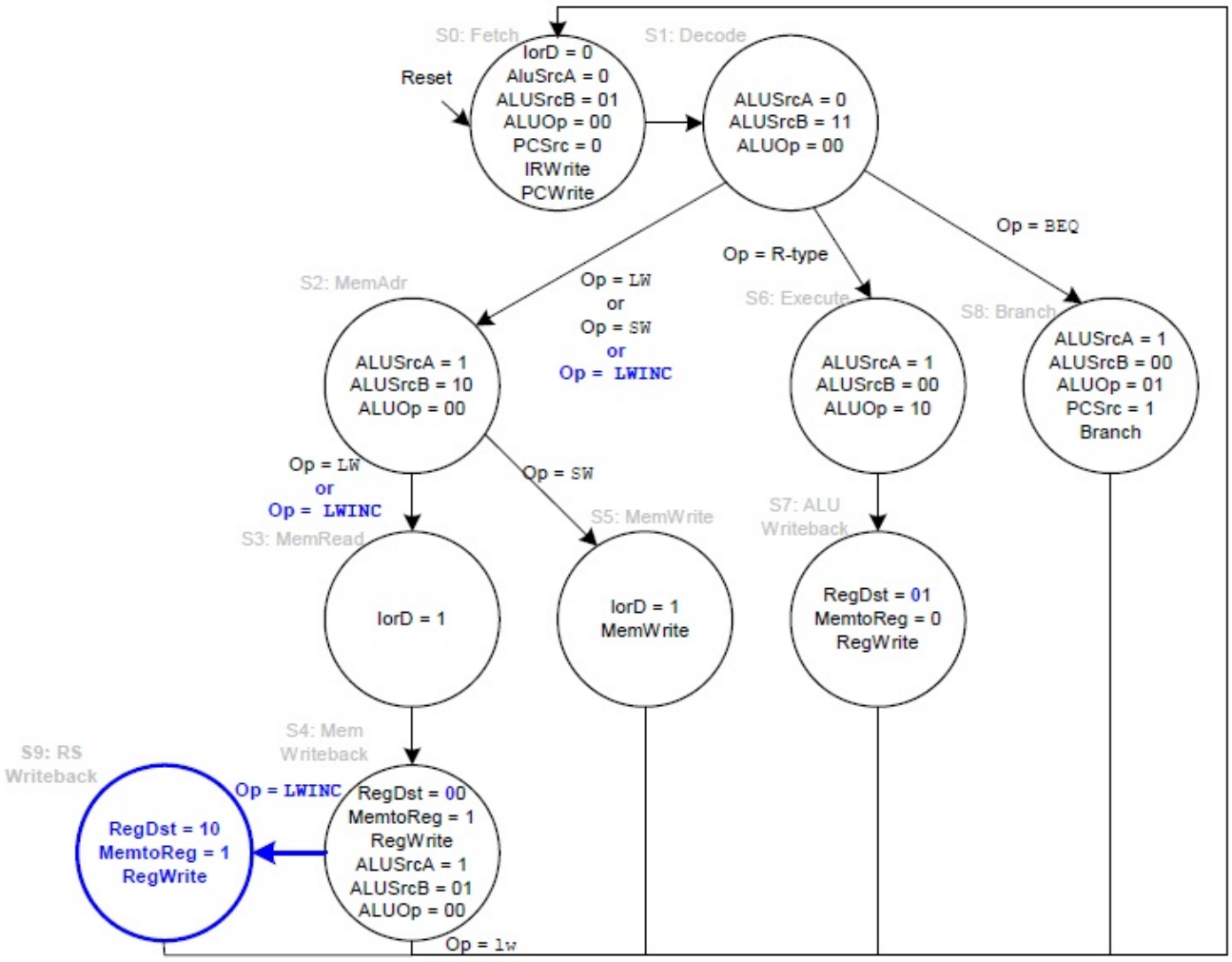
```
lw $rt, imm($rs)
addi $rs, $rs, 4
```



Hierzu muss der Multiplexer vor der Schreibadresse des Registersatzes erweitert werden, um auch $rs als Zielregister zu ermöglichen.




Nun muss auch das Steuerwerk erweitert werden:




## Aufgabe 13.3 Taktanzahl analysieren

Gegeben ist das folgende Codestück:

```
addi $s0, $zero, 5
while:
  beq $s0, $zero, done
  addi $so, $so, -1
  j while
done:
```

Geben Sie die Ausführungszeit auf einer Eintakt-Implementierung (Taktfrequenz = 100 MHz) und einer Mehrtakt-Implementierung (Taktfrequenz = 300 MHz) an.
Eintakt-Implementierung: Es sind 1 + 3*5 + 1 = 17 Befehle auszuführen. Bei 100 MHz benötigt dies 170 ns.
Mehrtakt-Implementierung: Es werden 4 + (3 + 4 + 3) * 5 + 3 = 57 Taktzyklen benötigt. Bei 300 MHz sind dies 190 ns.

## Aufgabe 13.4 Mehrtakt-Steuerwerk

Implementieren Sie das Steuerwerk des Mehrtakt-MIPS-Prozessors (Folie 7-63) in Verilog.

```
module maindec(input        clk, reset,
               input  [5:0] op,
               output       pcwrite, memwrite, irwrite, regwrite,
               output       alusrca, branch, iord, memtoreg, regdst,
               output [1:0] alusrcb,
```

```
              output       pcsrc,
              output [1:0] aluop
              );

parameter   FETCH   = 4'b0000;  // State 0
parameter   DECODE  = 4'b0001;  // State 1
parameter   MEMADR  = 4'b0010; // State 2
parameter   MEMRD   = 4'b0011; // State 3
parameter   MEMWB   = 4'b0100; // State 5
parameter   MEMWR   = 4'b0101; // State 5
parameter   RTYPEEX = 4'b0110; // State 6
parameter   RTYPEWB = 4'b0111; // State 7
parameter   BEQEX   = 4'b1000; // State 8

parameter   LW      = 6'b100011; // Opcode lw
parameter   SW      = 6'b101011; // Opcode sw
parameter   RTYPE   = 6'b000000; // Opcode R-type
parameter   BEQ     = 6'b000100; // Opcode beq
reg [3:0]  state, nextstate;
reg [13:0] controls;  //Ausgänge zusammengefasst

//nextstate
always @(posedge clk or posedge reset)
  if(reset) state <= FETCH;
  else state <= nextstate;

//Zustandsübergang
always @( * )
  case(state)
    FETCH:   nextstate <= DECODE;
    DECODE:  case(op)
               LW:      nextstate <= MEMADR;  //LW erkannt
               SW:      nextstate <= MEMADR;  //SW erkannt
               RTYPE:   nextstate <= RTYPEEX; //R-Typ Befehl
               BEQ:     nextstate <= BEQEX;   //BEQ
               default: nextstate <= FETCH;
             endcase
    //MEM-Befehle
    MEMADR:  case(op)
               LW:      nextstate <= MEMRD;
               SW:      nextstate <= MEMWR;
               default: nextstate <= FETCH;
             endcase
    MEMRD:   nextstate <= MEMWB;
    MEMWB:   nextstate <= FETCH;
    MEMWR:   nextstate <= FETCH;
    //R-Befehle
    RTYPEEX: nextstate <= RTYPEWB;
    RTYPEWB: nextstate <= FETCH;
    //BEQ
    BEQEX:   nextstate <= FETCH;
    default: nextstate <= FETCH;
  endcase

//Ausgaben zu einem Bus zusammenfassen
assign {pcwrite, memwrite, irwrite, regwrite,
        alusrca, branch, iord, memtoreg,
        regdst, alusrcb, pcsrc,
```

```
    aluop} = controls;

  //Ausgaben Steuersignale
  always @( * )
    case(state)
      FETCH:   controls <= 14'b1010_0000_0010_00;
      DECODE:  controls <= 14'b0000_0000_0110_00;
      MEMADR:  controls <= 14'b0000_1000_0100_00;
      MEMRD:   controls <= 14'b0000_0010_0000_00;
      MEMWB:   controls <= 14'b0000_0001_0000_00;
      MEMWR:   controls <= 14'b0100_0010_0000_00;
      RTYPEEX: controls <= 14'b0000_1000_0000_10;
      RTYPEWB: controls <= 14'b0001_0000_1000_00;
      BEQEX:   controls <= 14'b0000_1100_1001_01;
      default: controls <= 14'b0000_xxxx_xxxx_xx;
    endcase
endmodule
```

## Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism