SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Lösungsskizze 7

Aufgabe 7.1: Verilog: Tri-Port-RAM

```
module TriPortRAM(
    input clock,
    input[4:0] Aad, Bad, Xad,
    output[31:0] Aout, Bout,
    input[31:0] Xin,
    input we
);

reg[31:0] r[31:0];

assign Aout = r[Aad];
assign Bout = r[Bad];


always @(posedge clock) begin
    if (we) begin
        r[Xad] <= Xin;
    end
end

endmodule

module TestBench();

reg[31:0] x;
reg[4:0] addr;
wire[31:0] out;
reg clk, we;

TriPortRAM ram(
    .clock(clk),
    .Aad(addr),
    .Xad(addr),
    .Aout(out),
    .Xin(x),
    .we(we)
);

initial begin
    $dumpfile("out.vcd");
    $dumpvars();
```

```
        clk <= 0;
        we <= 1;
        addr <= 0;
        x <= 0;
        #1;
        clk <= 1;
        // Wir schreiben 0 in Register 0
        #1;
        x <= 42;
        // Der Wert 42 wird allerdings (noch) nicht geschrieben
        #1;
        clk <= 0;
        addr <= 1;
        #1;
        clk <= 1;
        // Wir Schreiben 42 in Register 1
        #1;
        clk <= 0;
        we <= 0;
        addr <= 0;
        // Wir lesen die 0;
        #1;
        addr <= 1;
        // Wir lesen die 42, obwohl die clock weiterhin auf 0 bleibt
        #1;
        $finish();
end
endmodule
```

Aufgabe 7.2: Dekodierer

(a) **Basisfall n=1:** C(D_1) = depth(D_1) = 1, weil es nur ein einziges NOT-Gatter gibt.

Fall n>1: *Tiefe:*

$$
\begin{aligned}
&depth(D_n) = depth(D_{n-1}) + 1 \\
&= depth(D_{n-2}) + 1 + 1 \\
&= ... \\
&= depth(D_{n-k}) + k * 1 \\
&= depth(D_1) + (n-1) * 1 \quad (n - k = 1 \Leftrightarrow k = n - 1) \\
&= 1 + n - 1 \\
&= n
\end{aligned}
$$

Beweis: per Induktion über n (Anzahl Bits)
Induktionsanfang: n=1
$n = 1 = depth(D_1)\surd$
Induktionsvoraussetzung:
Gelte für ein bel., festes $n \in \mathbb{N}$ die Aussage $depth(D_n) = n$.
Induktionsschritt: $n \to n + 1$

$$
\begin{aligned}
&depth(D_{n+1}) = depth(D_n) + 1 \\
&= n + 1\surd \quad (I.V.)
\end{aligned}
$$

■

Kosten:
$C(D_n) = C(D_{n-1}) + 1 + 2^n * 1$
$= C(D_{n-2}) + 1 + 2^{n-1} + 1 + 2^n$
$= ...$
$= C(D_{n-k}) + (\sum_{i=n-(k-1)}^{n} 2^i) + k$
$= C(D_1) + (\sum_{i=n-(n-1-1)}^{n} 2^i) + n - 1 \quad (n - k = 1 \Leftrightarrow k = n - 1)$
$= 1 + (\sum_{i=0}^{n} 2^i) - 2^1 - 2^0 + n - 1$
$= 2^{n+1} - 1 - 2 - 1 + n$
$= 2^{n+1} + n - 4$

Beweis: per Induktion über n (Anzahl Bits)
Induktionsanfang: n=1
$2^{1+1} + 1 - 4 = 5 - 4 = 1 = C(D_1)\surd$
Induktionsvoraussetzung:
Gelte für ein bel., festes $n \in \mathbb{N}$ die Aussage $C(D_n) = 2^{n+1} + n - 4$.
Induktionsschritt: $n \to n + 1$
$C(D_{n+1}) = C(D_n) + 1 + 2^{n+1}$
$= 2^{n+1} + n - 4 + 1 + 2^{n+1} \quad (I.V.)$
$= 2^{(n+1)+1} + (n + 1) - 4\surd$ ■

(b) Teile wie beim CSA die Bits in zwei Hälften und arbeite auf diesen rekursiv. Dabei sei $k = \lceil n/2 \rceil$. Siehe Abbildung 2[1]. Ein Korrektheitsbeweis findet sich im frei zugänglichen Sysbook der Vorlesung des SS13. (`http://www-wjp.cs.uni-saarland.de/lehre/vorlesung/info2/ss13/material/sysbook.pdf`)

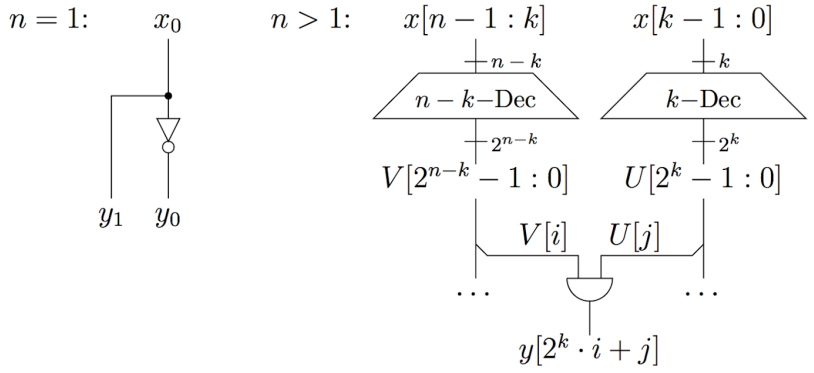

Abbildung 1: Rekursiver n-Bit Decoder

Kosten:
$C(D_n) = 2 * C(D_{n/2}) + 2^n$

Tiefe:
$depth(D_n) = 1 + depth(D_{n/2})$

Aufgabe 7.3: Befehlssatzarchitektur - MIPS

- Anhand der ersten 6 Bit des Befehls (der Opcode) erkennen wir, dass es sich um eine `ori` Instruktion (Seite 325 im MIPS32 Manual) handelt. Der Befehl wird wie folgt dekodiert:

$$\underbrace{001101}_{opcode}\ \underbrace{00101}_{rs}\ \underbrace{00100}_{rt}\ \underbrace{0000000001100100}_{imm}$$

[1] Entnommen aus W. J. Paul: Sysbook, SS 2013

Bei diesem Befehl ist das rs Register (hier `R5`) das Operandenregister und das rt Register (hier `R4`) das Zielregister. Die Immediate-Konstante repräsentiert 100_{10}.

Damit ist die gesuchte Instruktion `ori R4, R5, 100`.

- Wir kodieren `add R1, R2, R3` als:

$$\underbrace{000000}_{opcode}\underbrace{00010}_{rs=2}\underbrace{00011}_{rt=3}\underbrace{00001}_{rd=1}00000\underbrace{100000}_{funct}$$

und `ori R25, R12, 3637` als:

$$\underbrace{001101}_{opcode}\underbrace{01100}_{rs=12}\underbrace{11001}_{rt=25}\underbrace{0000111000110101}_{imm=3637}$$

- Das Programm schreibt mit den ersten beiden Instruktionen den Wert `0x01020304` in `R2`. Dieses Wort wird dann an Stelle 4 ($= 4+$`R0`) gespeichert. Die letzte Instruktion lädt das Byte, das an Stelle 4 im Speicher steht in `R3`. Welchen Wert dieses Byte hat, hängt von der Endianness des Systems ab:
  - im Falle einer Big Endian Architektur (wie in der Vorlesung festgelegt) `0x01`,
  - für eine Little Endian Architektur `0x04`.

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Solution Sketch 7

Problem 7.1: Verilog: Tri-Port RAM

```
module TriPortRAM(
    input clock,
    input[4:0] Aad, Bad, Xad,
    output[31:0] Aout, Bout,
    input[31:0] Xin,
    input we
);

reg[31:0] r[31:0];

assign Aout = r[Aad];
assign Bout = r[Bad];


always @(posedge clock) begin
    if (we) begin
        r[Xad] <= Xin;
    end
end

endmodule

module TestBench();

reg[31:0] x;
reg[4:0] addr;
wire[31:0] out;
reg clk, we;

TriPortRAM ram(
    .clock(clk),
    .Aad(addr),
    .Xad(addr),
    .Aout(out),
    .Xin(x),
    .we(we)
);

initial begin
    $dumpfile("out.vcd");
    $dumpvars();
```

```
        clk <= 0;
        we <= 1;
        addr <= 0;
        x <= 0;
        #1;
        clk <= 1;
        // We write 0 to register 0
        #1;
        x <= 42;
        // However, the value 42 is not written (yet)
        #1;
        clk <= 0;
        addr <= 1;
        #1;
        clk <= 1;
        // We write 42 to register 1
        #1;
        clk <= 0;
        we <= 0;
        addr <= 0;
        // We read the 0
        #1;
        addr <= 1;
        // We read the 42, even though the clock remains at 0
        #1;
        $finish();
end
endmodule
```

Problem 7.2: Decoder

(a) **Base case n=1:** C(D_1) = depth(D_1) = 1, because there is only one NOT gate.

Case n>1: *Depth:*

$$
\begin{aligned}
&depth(D_n) = depth(D_{n-1}) + 1 \\
&= depth(D_{n-2}) + 1 + 1 \\
&= ... \\
&= depth(D_{n-k}) + k * 1 \\
&= depth(D_1) + (n-1) * 1 \quad (n - k = 1 \Leftrightarrow k = n - 1) \\
&= 1 + n - 1 \\
&= n
\end{aligned}
$$

Proof: by induction on n (number of bits)
Base case: n=1
$n = 1 = depth(D_1)\surd$
Induction hypothesis (IH):
The claim $depth(D_n) = n$ holds for an arbitrary but fixed $n \in \mathbb{N}$.
Induction step: $n \to n + 1$
$depth(D_{n+1}) = depth(D_n) + 1$
$= n + 1\surd \quad (IH)$ ■

Cost:
$C(D_n) = C(D_{n-1}) + 1 + 2^n * 1$
$= C(D_{n-2}) + 1 + 2^{n-1} + 1 + 2^n$
$= ...$
$= C(D_{n-k}) + (\sum_{i=n-(k-1)}^{n} 2^i) + k$
$= C(D_1) + (\sum_{i=n-(n-1-1)}^{n} 2^i) + n - 1 \quad (n-k=1 \Leftrightarrow k = n-1)$
$= 1 + (\sum_{i=0}^{n} 2^i) - 2^1 - 2^0 + n - 1$
$= 2^{n+1} - 1 - 2 - 1 + n$
$= 2^{n+1} + n - 4$

Proof: by induction on n (number of bits)
Base case: n=1
$2^{1+1} + 1 - 4 = 5 - 4 = 1 = C(D_1)\surd$
Induction hypothesis (IH):
The claim $C(D_n) = 2^{n+1} + n - 4$ holds for an arbitrary but fixed $n \in \mathbb{N}$.
Induction step: $n \to n+1$
$C(D_{n+1}) = C(D_n) + 1 + 2^{n+1}$
$= 2^{n+1} + n - 4 + 1 + 2^{n+1} \quad (IH)$
$= 2^{(n+1)+1} + (n+1) - 4\surd$ ■

(b) As with the CSA, partition the bits into two halves and work on them recursively. Let $k = \lceil n/2 \rceil$. See Figure 2[2]. A proof of correctness can be found in the freely available Sysbook of the SS13 course. http://www-wjp.cs.uni-saarland.de/lehre/vorlesung/info2/ss13/material/sysbook.pdf.

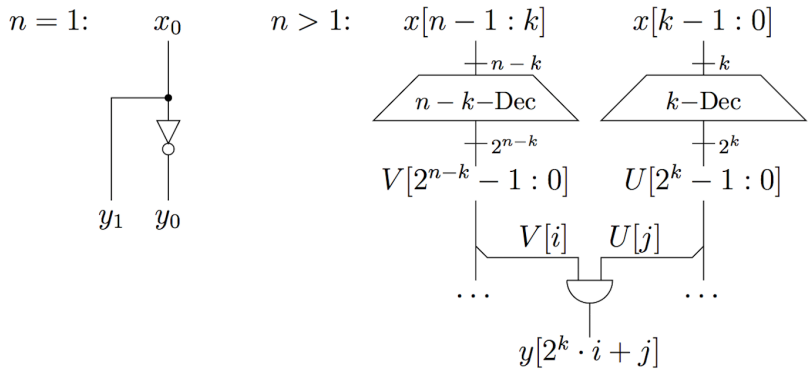

Figure 2: Recursive n-bit decoder

Cost:
$C(D_n) = 2 * C(D_{n/2}) + 2^n$

Depth:
$depth(D_n) = 1 + depth(D_{n/2})$

Problem 7.3: Instruction Set Architecture - MIPS

- From the first 6 bits of the instruction (the opcode), we can see that it is an `ori` instruction (page 325 in the MIPS32 manual). The instruction is decoded as follows:

$$\underbrace{001101}_{opcode}\ \underbrace{00101}_{rs}\ \underbrace{00100}_{rt}\ \underbrace{0000000001100100}_{imm}$$

[2]Taken from W. J. Paul: Sysbook, SS 2013

In this instruction, the rs register (here `R5`) is the operand register, and the rt register (here `R4`) is the destination register. The immediate constant represents 100_{10}.

Thus, the decoded instruction is `ori R4, R5, 100`.

- We encode `add R1, R2, R3` as:

$$\underbrace{000000}_{opcode}\underbrace{00010}_{rs=2}\underbrace{00011}_{rt=3}\underbrace{00001}_{rd=1}00000\underbrace{100000}_{funct}$$

and `ori R25, R12, 3637` as:

$$\underbrace{001101}_{opcode}\underbrace{01100}_{rs=12}\underbrace{11001}_{rt=25}\underbrace{0000111000110101}_{imm=3637}$$

- With the first two instructions, the program writes the value `0x01020304` to `R2`. This word is then stored at location 4 ($= 4 +$ `R0`). The last instruction loads the byte that is at location 4 in memory into `R3`. The value of this byte depends on the endianness of the system:
  - in the case of a Big Endian architecture (as specified in the lecture): `0x01`,
  - for a Little Endian architecture: `0x04`.