

Note: The Elements of Computing Systems

September 10, 2020

Preface

This is my study note for The Elements of Computing Systems (ISBN: 978-0262640688) by Noam Nisan and Shimon Schocken.

Chapter 1

Boolean Logic

Boolean Algebra

$$\begin{aligned}
 x \text{ or } y &= \bar{x}y + x\bar{y} + xy \\
 &= \bar{x}y + x(y + \bar{y}) \\
 &= x + \bar{x}y \\
 &= x + \overline{x + \bar{y}} \\
 &= \overline{\overline{x + \bar{y}}} \\
 &= \overline{\bar{x}(x + \bar{y})} \quad \text{where } \overline{x + \bar{y}} = \bar{x}\bar{y} \\
 &= \overline{x\bar{x} + \bar{x}\bar{y}} \\
 &= \overline{\bar{x}\bar{y}} \\
 &= \overline{\overline{x + y}} \\
 &= x + y
 \end{aligned}$$

$$\begin{aligned}
 \text{if } y \text{ then } x &= \bar{x}\bar{y} + x\bar{y} + xy \\
 &= \bar{x}\bar{y} + x(y + \bar{y}) \\
 &= x + \bar{x}\bar{y} \\
 &= x + \overline{x + y} \\
 &= \overline{\overline{x + y}} \\
 &= \overline{\bar{x}(x + y)} \\
 &= \overline{x\bar{x} + \bar{x}y} \\
 &= \overline{\bar{x}y} \\
 &= x + \bar{y}
 \end{aligned}$$

$$\begin{aligned}
 \text{if } x \text{ then } y &= \bar{x}\bar{y} + \bar{x}y + xy \\
 &= \bar{x}\bar{y} + (x + \bar{x})y \\
 &= \bar{x}\bar{y} + y \\
 &= \overline{x + y} + y \\
 &= \overline{\overline{x + y} + y} \\
 &= \overline{(x + y)\bar{y}} \\
 &= \overline{x\bar{y} + y\bar{y}} \\
 &= \overline{\bar{x}\bar{y}} \\
 &= \bar{x} + y
 \end{aligned}$$

$$\begin{aligned}
 x \text{ nand } y &= \bar{x}\bar{y} + \bar{x}y + x\bar{y} \\
 &= \bar{y}(x + \bar{x}) + \bar{x}y \\
 &= \bar{y} + \bar{x}y \\
 &= \bar{x} + \bar{y} \quad \text{where } x + y = x + \bar{x}y \\
 &= \overline{\overline{\bar{x} + \bar{y}}} \\
 &= \overline{\bar{x}\bar{y}}
 \end{aligned}$$

Chapter 5

Computer Architecture

5.1 Memory

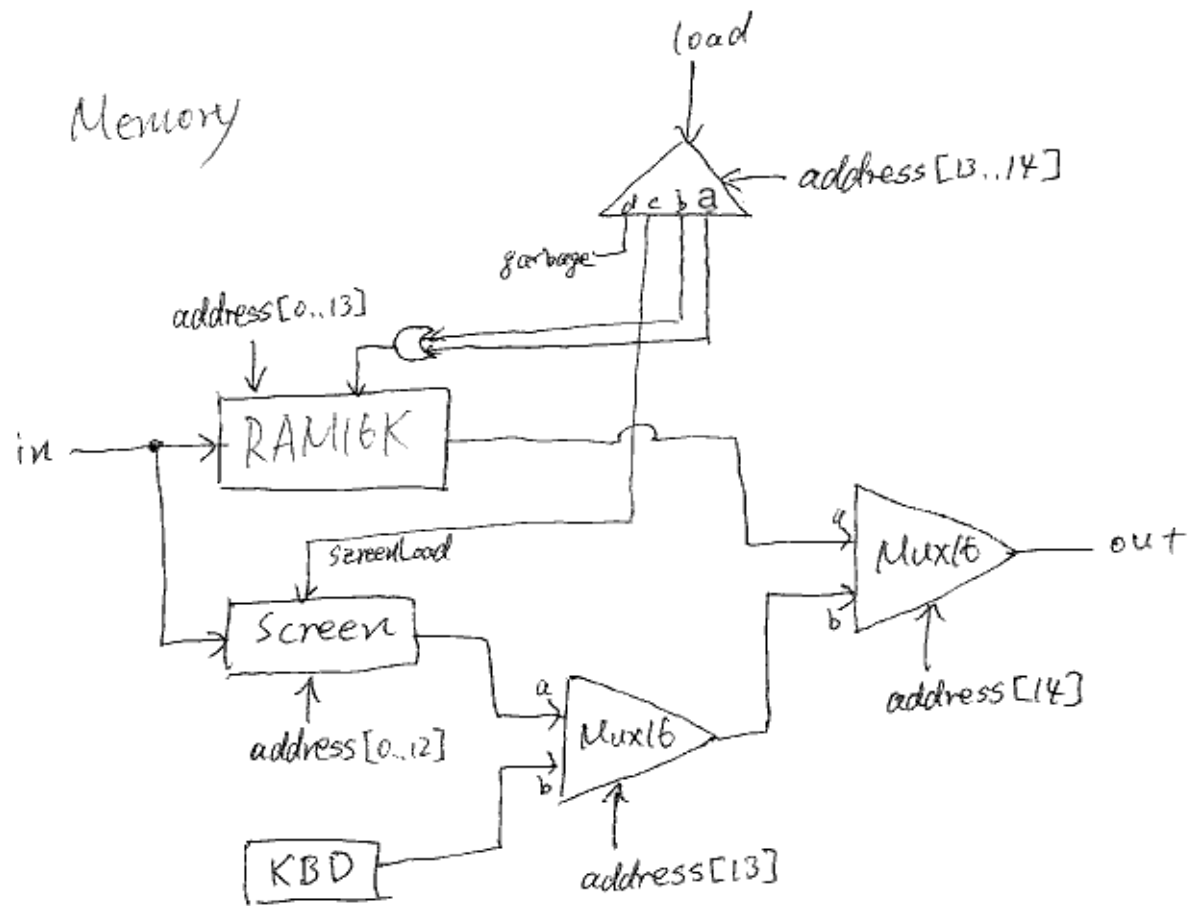


Figure 5.1: memory

5.2 CPU

C-instruction = $111ac_1c_2c_3c_4c_5c_6d_1d_2d_3j_1j_2j_3$

- d_1 : destination A
- d_2 : destination D
- d_3 : destination M

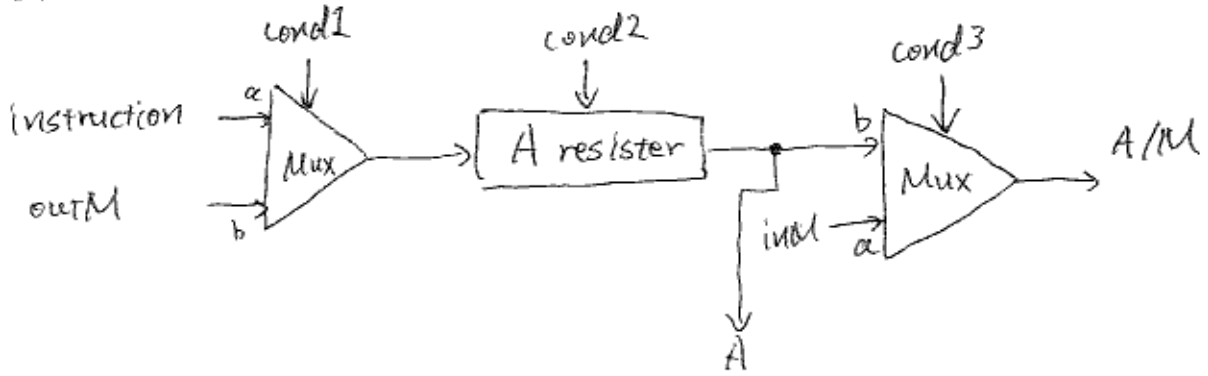


Figure 5.2: memory

$\text{cond1} = \text{instruction}[15]$
 $\text{cond2} = (\text{not } \text{instruction}[15]) \text{ or } (\text{instruction}[15] \text{ and } \text{instruction}[5])$
 $\text{cond3} = \text{instruction}[15] \text{ and } \text{not } \text{instruction}[12]$

- $\text{instruction}[15]$: opcode
- $\text{instruction}[12]$: C-instruction's a . If a is 1, comp includes A, otherwise, comp includes M.
- $\text{instruction}[5]$: destination A.

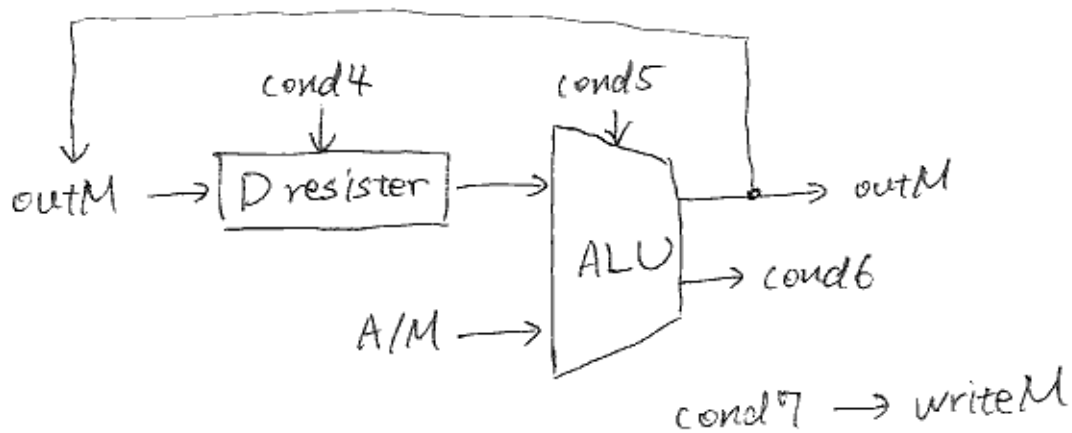


Figure 5.3: memory

$\text{cond4} = \text{instruction}[15] \text{ and } \text{instruction}[4]$

$\text{cond5} = \begin{cases} \text{zx} = \text{instruction}[11] = c_1 \\ \text{nx} = \text{instruction}[10] = c_2 \\ \text{zy} = \text{instruction}[9] = c_3 \\ \text{ny} = \text{instruction}[8] = c_4 \\ \text{f} = \text{instruction}[7] = c_5 \\ \text{no} = \text{instruction}[6] = c_6 \end{cases}$

$\text{cond6} = (\text{zr}, \text{ng})$

$\text{cond7} = \text{instruction}[15] \text{ and } \text{instruction}[3]$

- $\text{instruction}[3]$: d_3 , destination M

- instruction[4]: d_2 , destination D

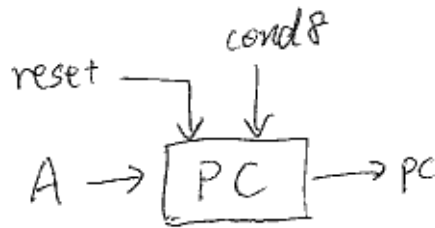


Figure 5.4: memory

$$\begin{aligned}
 \text{cond8} = & \overline{\text{zr}} \cdot \overline{\text{ng}} \cdot \overline{j_1} \cdot \overline{j_2} \cdot \overline{j_3} & (\text{JGT}) \\
 & + \overline{\text{zr}} \cdot \overline{j_1} \cdot j_2 \cdot j_3 & (\text{JEQ}) \\
 & + \overline{\text{ng}} \cdot \overline{j_1} \cdot j_2 \cdot j_3 & (\text{JGE}) \\
 & + \text{ng} \cdot j_1 \cdot \overline{j_2} \cdot \overline{j_3} & (\text{JLT}) \\
 & + \overline{\text{zr}} \cdot j_1 \cdot \overline{j_2} \cdot j_3 & (\text{JNE}) \\
 & + (\text{zr} + \text{ng}) \cdot j_1 \cdot j_2 \cdot \overline{j_3} & (\text{JLE}) \\
 & + j_1 \cdot j_2 \cdot j_3 & (\text{JMP})
 \end{aligned}$$

Chapter 7

Virtual Machine I: Stack Arithmetic

7.1 Arithmetic

add (sub) \Rightarrow {

- // SP--
- @SP
- M=M-1
-
- // D = y
- A=M
- D=M
-
- // SP--
- @SP
- M=M-1
-
- // *SP = y + x (add) / x - y (sub)
- A=M
- M=D+M (add) / M=M-D (sub)
-
- // SP++
- @SP
- M=M+1

neg, not \Rightarrow {

- // SP--
- @SP
- @M=M-1
-
- // -M
- A=M
- M=-M (neg) / M=!M (not)
-
- // SP++
- @SP
- M=M+1

```

eq, gt, lt ⇒ {
    // SP--
    @SP
    M=M-1

    // D = y
    D=M

    // SP--
    @SP
    M=M-1

    // x - y
    A = M
    D=M-D

    // if condition then -1 else 0 end
    @then
    D;jEQ (eq), D;jGT (gt), D;jLT (lt)
    @SP
    A=M
    M=0
    @end
    0;JMP
    (then)
    @SP
    A=M
    M=-1
    (end)

    // SP++
    @SP
    M=M+1

```

$$\text{and, or} \Rightarrow \left\{ \begin{array}{l} // \text{ SP}++ \\ @SP \\ M=M-1 \\ \\ // \text{ D} = y \\ A=M \\ D=M \\ \\ // \text{ SP}-- \\ @SP \\ M=M-1 \\ \\ // *SP = y\&x \\ A=M \\ M=D\&M \\ \\ // \text{ SP}++ \\ @SP \\ M=M+1 \end{array} \right.$$

7.2 logical command

$$\begin{aligned} \text{push constant } i &\Rightarrow \left\{ \begin{array}{l} *SP = i \\ SP++ \end{array} \right. \Rightarrow \left\{ \begin{array}{l} // *SP=i \\ @i \\ D=A \\ @SP \\ A=M \\ M=D \\ \\ // \text{ SP}++ \\ @SP \\ M=M+1 \end{array} \right. \\ \\ \text{push segment } i &\Rightarrow \left\{ \begin{array}{l} \text{addr} = \text{SEG} + i \\ *SP = *addr \\ SP++ \end{array} \right. \Rightarrow \left\{ \begin{array}{l} // \text{ addr} = \text{SEG} + i \\ @SEG \\ D=M \\ @i \\ A=D+A \\ \\ // *addr \\ D=M \\ \\ // *SP = *addr \\ @SP \\ A=M \\ M=D \\ \\ // \text{ SP}++ \\ @SP \\ M=M+1 \end{array} \right. \end{aligned}$$

where `segment` = `local`, `argument`, `this`, `that`.

$$\begin{aligned} \text{push pointer } i \Rightarrow & \left\{ \begin{array}{l} @THIS \text{ // or } @R3 \\ D=M \\ @i \\ A=D+A \\ D=M \\ @SP \\ A=M \\ M=D \\ @SP \\ M=M+1 \end{array} \right. \\ \\ \text{push static } i \Rightarrow & \left\{ \begin{array}{l} @Xxx.i \\ D=M \\ @SP \\ A=M \\ M=D \\ @SP \\ M=M+1 \end{array} \right. \end{aligned}$$

where this vm program name is `Xxx.vm`.

$$\begin{aligned} \text{pop segment } i \Rightarrow & \left\{ \begin{array}{l} \text{addr} = \text{SEG} + i \\ \text{SP}-- \\ *addr = *SP \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \text{SEG} = \text{SEG} + i \\ \text{SP}-- \\ *SEG = *SP \\ \text{SEG} = \text{SEG} - i \end{array} \right. \Rightarrow \left\{ \begin{array}{l} // \text{ SEG}=\text{SEG}+i \\ @SEG \\ D=M \\ @i \\ D=D+A \\ @SEG \\ M=D \\ \\ // \text{ SP}-- \\ @SP \\ M=M-1 \\ \\ // *SP \\ A=M \\ D=M \\ \\ // *SEG = *SP \\ @SEG \\ A=M \\ M=D \\ \\ // \text{ SEG}=\text{SEG}-i \\ @i \\ D=A \\ @SEG \\ M=M-D \end{array} \right. \end{aligned}$$

where `segment` = `local`, `argument`, `this`, `that`.

`pop pointer i` is equivalent to `pop this i` where $i = 0, 1$.

$$\text{pop static } i \Rightarrow \left\{ \begin{array}{l} // \text{ SP--} \\ @SP \\ M=M-1 \\ \\ // *SP \\ A=M \\ D=M \\ \\ @Xxx.i \\ M=D \end{array} \right.$$