

# Note: The Elements of Computing Systems

September 21, 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

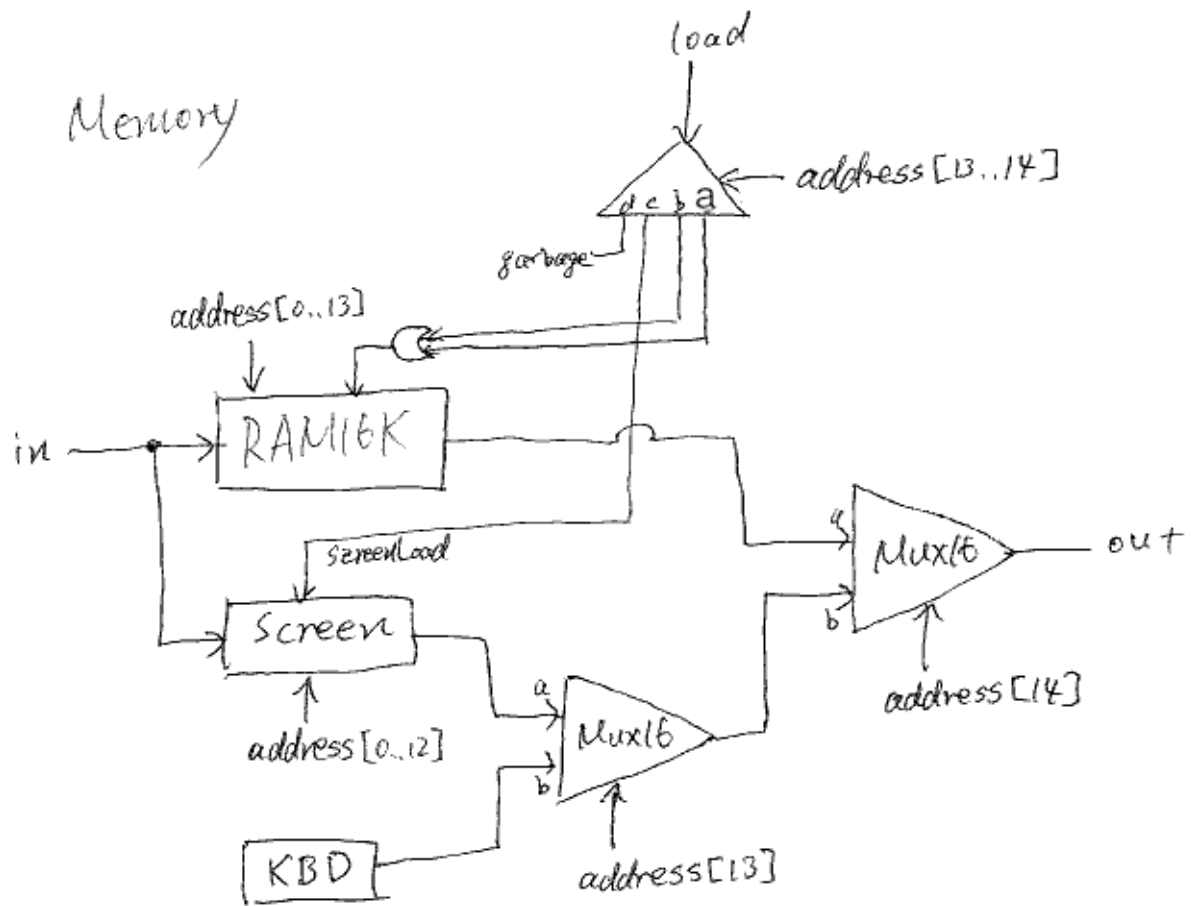


Figure 5.1: memory

### 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

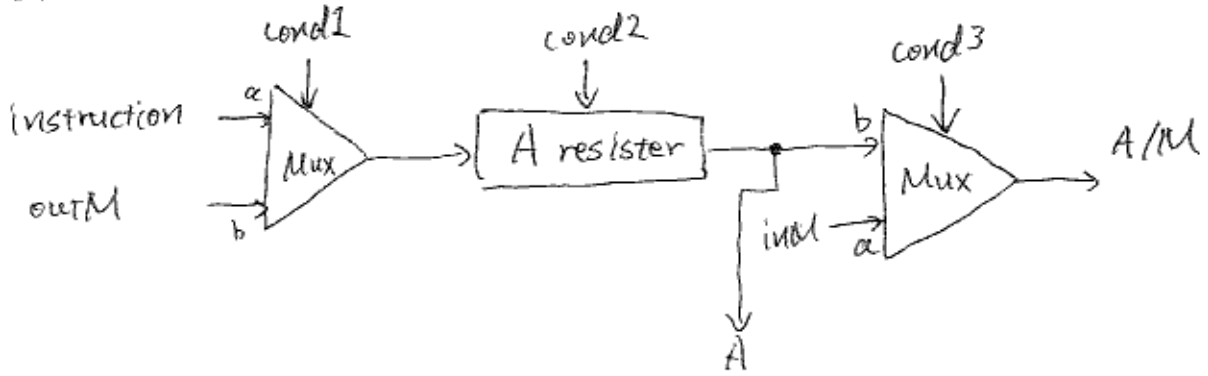


Figure 5.2: memory

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

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

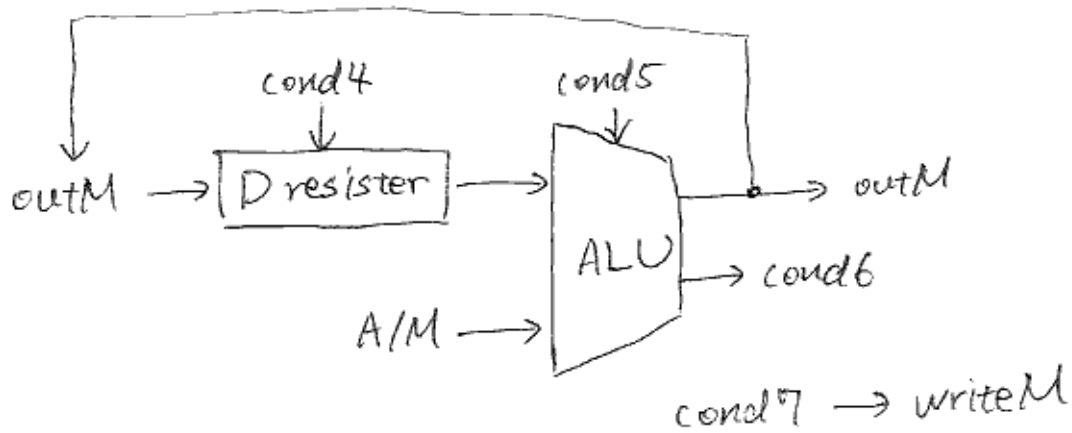


Figure 5.3: memory

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

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

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

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

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



- instruction[4]:  $d_2$ , destination D

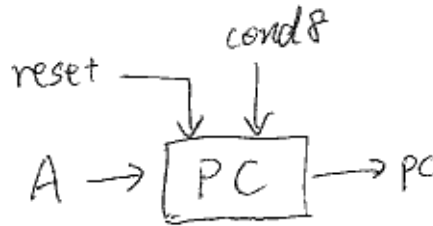


Figure 5.4: memory

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



## 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
    A=M
    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

```

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

## 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`.

$$\text{push pointer } i \Rightarrow \left\{ \begin{array}{l} *SP = *(R3 + i) \end{array} \right. \Rightarrow \left\{ \begin{array}{l} // *(R3 + i) \\ @R(3 + i) \\ D=M \\ // *SP = *(R3 + i) \\ @SP \\ A=M \\ M=D \end{array} \right.$$

$$\text{push temp } i \Rightarrow \left\{ \begin{array}{l} *SP = *(R5 + i) \end{array} \right. \Rightarrow \left\{ \begin{array}{l} // *(R5 + i) \\ @R(5 + i) \\ D=M \\ // *SP = *(R5 + i) \\ @SP \\ A=M \\ M=D \end{array} \right.$$

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

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

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

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

$$\begin{aligned}
\text{pop pointer } i \Rightarrow & \begin{cases} \text{SP}-- \\ * \text{SP} \\ *(R3 + i) = * \text{SP} \end{cases} \Rightarrow \begin{cases} // \text{ SP}-- \\ @ \text{SP} \\ \text{M}=\text{M}-1 \\ \\ // * \text{SP} \\ \text{A}=\text{M} \\ \text{D}=\text{M} \\ \\ @R(3 + i) \\ \text{M}=\text{D} \end{cases} \\
\text{pop temp } i \Rightarrow & \begin{cases} \text{SP}-- \\ * \text{SP} \\ *(R5 + i) = * \text{SP} \end{cases} \Rightarrow \begin{cases} // \text{ SP}-- \\ @ \text{SP} \\ \text{M}=\text{M}-1 \\ \\ // * \text{SP} \\ \text{A}=\text{M} \\ \text{D}=\text{M} \\ \\ @R(5 + i) \\ \text{M}=\text{D} \end{cases}
\end{aligned}$$

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



## Chapter 11

# Compiler II: Code Generation

```
label symbol
  goto symbol
if-goto symbol
function functionName nLocals
  call functionName nArgs
```

### Expression

#### Integer constant

```
cgen(n) = push constant n
```

where **n** is an integer.

#### String constant

```
cgen(str) = push constant length(str)
           call String.new 1
           push constant ASCII(str1)
           call String.appendChar 2
           push constant ASCII(str2)
           call String.appendChar 2
           :
           push constant ASCII(strn)
           call String.appendChar 2
```

where **str** is a string constant, **stri** is *i*th character of **str**, ASCII(**x**) is ASCII number of char **x** and **n** is the length of **str**.

#### Keyword constant

```
cgen(true) = push constant 0
            neg
```

```
cgen(false) = push constant 0
```

```
cgen(null) = push constant 0
```

```
cgen(this) = push pointer 0
```

## Variable

$$\text{cgen}(x) = \text{push segment } i$$

where  $x$  is a variable and `segment i` is related to  $x$ .

$$\begin{aligned} \text{cgen}(\text{arr}[\text{expr}] ) &= \text{cgen}(\text{expr} ) \\ &\quad \text{push segment } i \\ &\quad \text{add} \\ &\quad \text{pop pointer } 1 \\ &\quad \text{push that } 0 \end{aligned}$$

where `arr` is an Array and `segment i` is related to `arr`.

## Subroutine Call

$$\begin{aligned} \text{cgen}(\text{subroutineName}(\text{expressionList})) &= \text{push pointer } 0 \\ &\quad \text{cgen}(\text{expressionList}) \\ &\quad \text{call } \text{ClassName.subroutineName } z \end{aligned}$$

where `ClassName` is the name of class that `subroutineName` belong to and  $z = nArgs + 1$ .

$$\begin{aligned} \text{cgen}(\text{obj.subroutineName}(\text{expressionList})) &= \text{cgen}(\text{obj}) \\ &\quad \text{cgen}(\text{expressionList}) \\ &\quad \text{call } \text{ClassName.subroutineName } z \end{aligned}$$

where `ClassName` is the class name of `obj` and  $z = nArgs + 1$ .

$$\begin{aligned} \text{cgen}(\text{className.subroutineName}(\text{expressionList})) &= \text{cgen}(\text{expressionList}) \\ &\quad \text{call } \text{ClassName.subroutineName } nArgs \end{aligned}$$

$$\begin{aligned} \text{cgen}(\text{expressionList}) &= \text{cgen}(\text{expr1}, \text{expr2}, \dots, \text{exprn}) \\ &= \text{cgen}(\text{expr1}) \\ &\quad \vdots \\ &\quad \text{cgen}(\text{exprn}) \end{aligned}$$

## Parenthesis

$$\text{cgen}\{ (\text{expr} ) \} = \text{cgen}(\text{expr} )$$

## Unary Operator

$$\begin{aligned} \text{cgen}(\text{op term} ) &= \text{cgen}(\text{term}) \\ &\quad \text{op} \end{aligned}$$

## Operator

$$\begin{aligned} \text{cgen}(\text{term1 op term2} ) &= \text{cgen}(\text{term1}) \\ &\quad \text{cgen}(\text{term2}) \\ &\quad \text{op} \end{aligned}$$

## Statement

### let

```
cgen(let varName = expr ; ) = cgen(expr)
                             pop segment i
```

where `segment i` is related to `varName`.

```
cgen( let arr[expr1] = expr2 ; ) = cgen(arr)
                                cgen(expr1)
                                add
                                cgen(expr2)
                                pop temp 0
                                pop pointer 1
                                push temp 0
                                pop that 0
```

### if

```
cgen( if ( expr ) { statements } ) = cgen( expr )
                                if-goto label
                                cgen( statements )
                                label label
```

```
cgen( if ( expr ) { statements1 } else { statements2 } ) = cgen( expr )
                                                            if-goto else
                                                            cgen( statements1 )
                                                            goto end
                                                            label else
                                                            cgen( statements2 )
                                                            label end
```

### while

```
cgen( while (expr) { statements } ) = label begin
                                cgen( expr )
                                if-goto end
                                cgen( statements )
                                goto begin
                                label end
```

### do

```
cgen( do subroutineCall ; ) = cgen( subroutineCall )
                             pop temp 0
```

**return**

```
cgen( return ; ) = push constant 0
                    return
```

```
cgen( return expr; ) = cgen(expr)
                    return
```

## Subroutine Declaration

$$\text{cgen} \left( \begin{array}{l} \text{constructor className subroutineName(parameters) \{ } \\ \text{varDec*} \\ \text{statements} \\ \} \end{array} \right)$$

```
= function className.subroutineName nLocals
    push constant nFields
    call Memory.alloc 1
    pop pointer 0
    cgen( statements )
```

where `className` is the class that `subroutineName` belong to and `nFields` is the number of field variable in the class.

$$\text{cgen} \left( \begin{array}{l} \text{function type subroutineName(parameters) \{ } \\ \text{varDec*} \\ \text{statements} \\ \} \end{array} \right)$$

```
= function className.subroutineName nLocals
    cgen( statements )
```

where `className` is the class that `subroutineName` belong to.

$$\text{cgen} \left( \begin{array}{l} \text{method type subroutineName(parameters) \{ } \\ \text{varDec*} \\ \text{statements} \\ \} \end{array} \right)$$

```
= function className.subroutineName nLocals
    push argument 0
    pop pointer 0
    cgen( statements )
```

where `className` is the class that `subroutineName` belong to.