Computer Organization

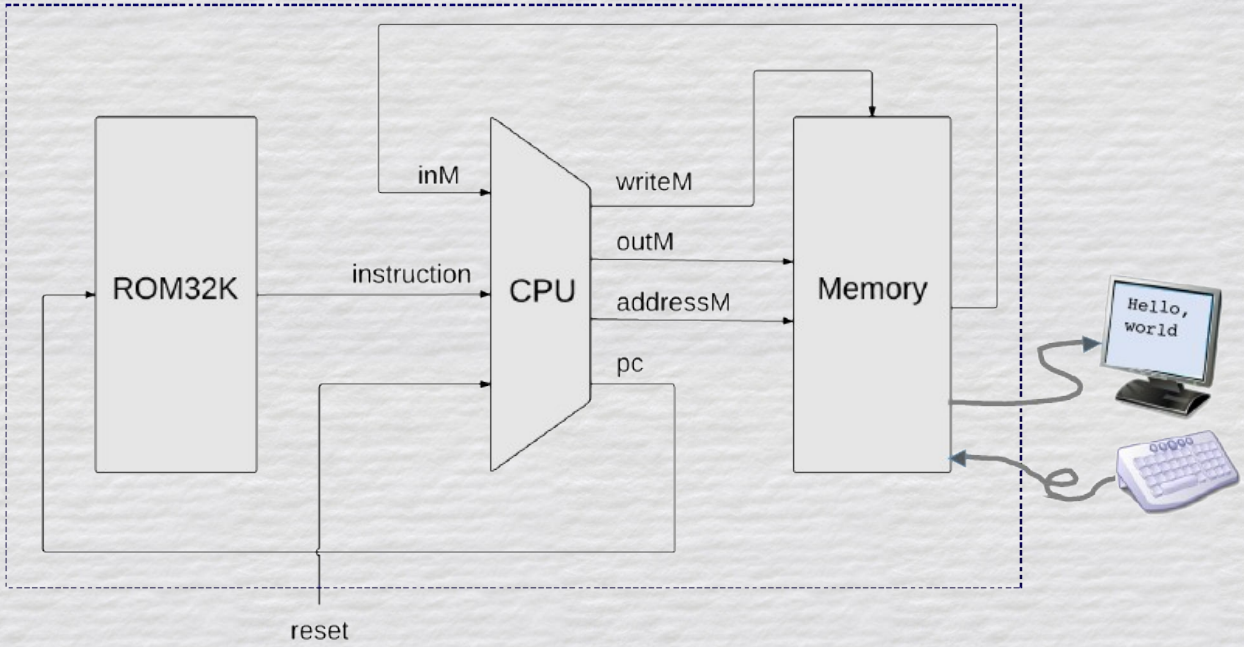
Assembling the Computer

Computer Implementation

- The HACK computer consists of 3 major components
  - ROM
  - CPU
  - RAM

- We can quite easily implement this using our previous chapters and a few provided chips

Computer Implementation – cont.

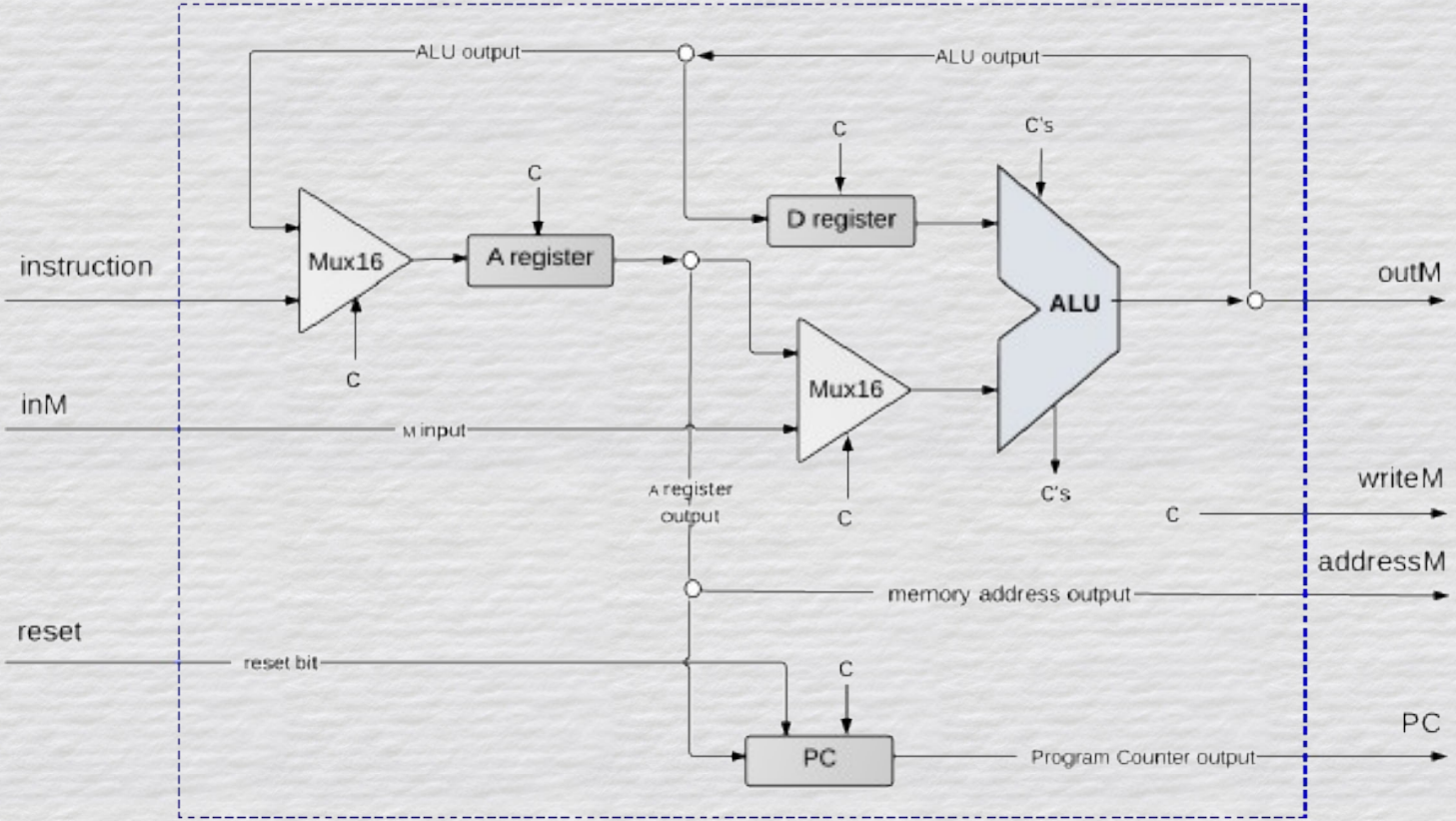



```
17  CHIP Computer {
18
19      IN reset;
20
21      PARTS:
22      ROM32K(address=pc,out=instruction);
23
24      CPU(instruction=instruction,reset=reset,
25          inM=inM,outM=outM,writeM=writeM,
26          addressM=addressM,pc=pc);
27
28      Memory(in=outM,load=writeM,address=addressM,
29          out=inM);
30  }
```

CPU

- The CPU is by far the most difficult component to implement
- It primarily consists of the ALU, A/D/PC Registers, and several basic gates for the logic reasons
- The best method to implement the chip is to break it down in several steps

Computer Implementation – cont.



Instruction Bits

- Before we can begin implementing the CPU, we need to take a look at our main input: **The Instruction**
- The Instruction is a 16-bit input that is either an A or a C instruction
- We can use the previous chapter to understand how these instructions are setup

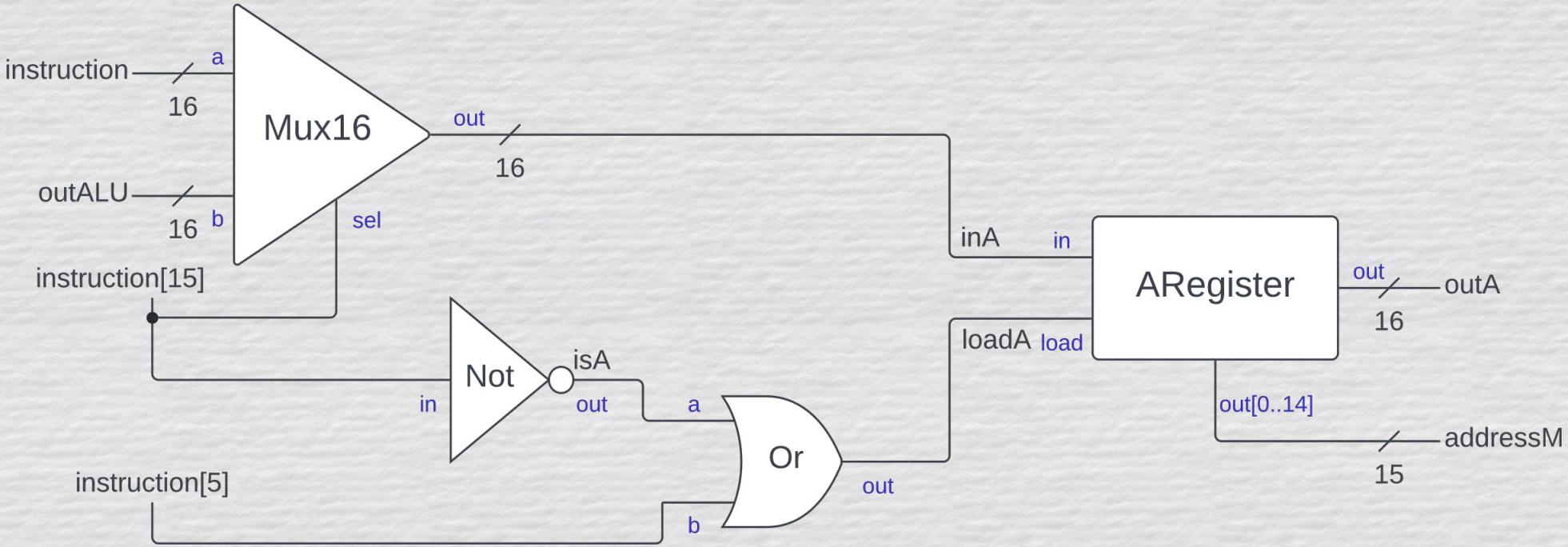
Instruction Bits – cont.

```
42      /*
43      instruction bits:
44          instruction[15] == Op-code bit (determines A or C instruction)
45          instruction[14] == Unused bit during C instructions
46          instruction[13] == Unused bit during C instructions
47          instruction[12] == A bit (determines if M register is used during ALU operation)
48          instruction[11] == ALU zx bit (c1 bit)
49          instruction[10] == ALU nx bit (c2 bit)
50          instruction[9] == ALU zy bit (c3 bit)
51          instruction[8] == ALU ny bit (c4 bit)
52          instruction[7] == ALU f bit (c5 bit)
53          instruction[6] == ALU no bit (c6 bit)
54          instruction[5] == A register bit (d1 bit)
55          instruction[4] == D register bit (d2 bit)
56          instruction[3] == M register bit (d3 bit)
57          instruction[2] == LT bit (j1 bit)
58          instruction[1] == EQ bit (j2 bit)
59          instruction[0] == GT bit (j3 bit)
60      */
```

Implementing the CPU

- Now that we have looked at the overview and the instruction bits, we can begin implementing the various components
  - A-Register
  - D-Register
  - M-Register
  - ALU
  - Jump bits
  - PC Register

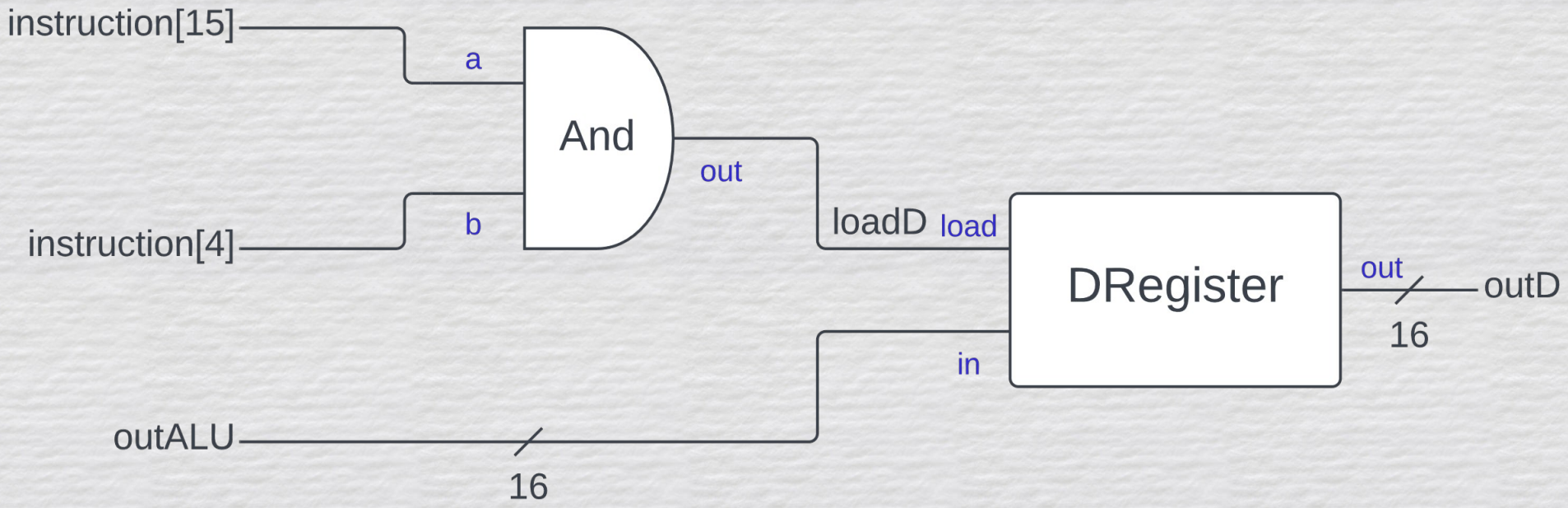
A-Register


instruction
16
a
Mux16
out
16
outALU
16
b
sel
instruction[15]
inA
in
ARegister
out
outA
16
loadA
load
Not
isA
in
out
a
Or
out
out[0..14]
addressM
15
instruction[5]
b

A-Register – Code

```
62      //=============================================================================================
63      // A-Register
64      /*
65          Use the Op-Code to choose between a constant (A-Instruction)
66          and the ALU's output (C-Instruction) for the input to the
67          A-Register
68      */
69      Mux16(a=, b=, sel=, out=);
70
71      // Not the Op-Code to determine if an A-Instruction is being processed
72      Not(in=, out=);
73
74      /*
75          When !instruction[15] == 1, it is @value which means A should load a value
76          When instruction[5] == 1, it is A=outALU which means A should load a value
77      */
78      Or(a=, b=, out=);
79
80      /*
81          Loads a new input to the A-Register if above code dictates it to
82          Outputs both to A-Register and sets the memory address
83      */
84      ARegister(in=, load=, out=, out[0..14]=);
```

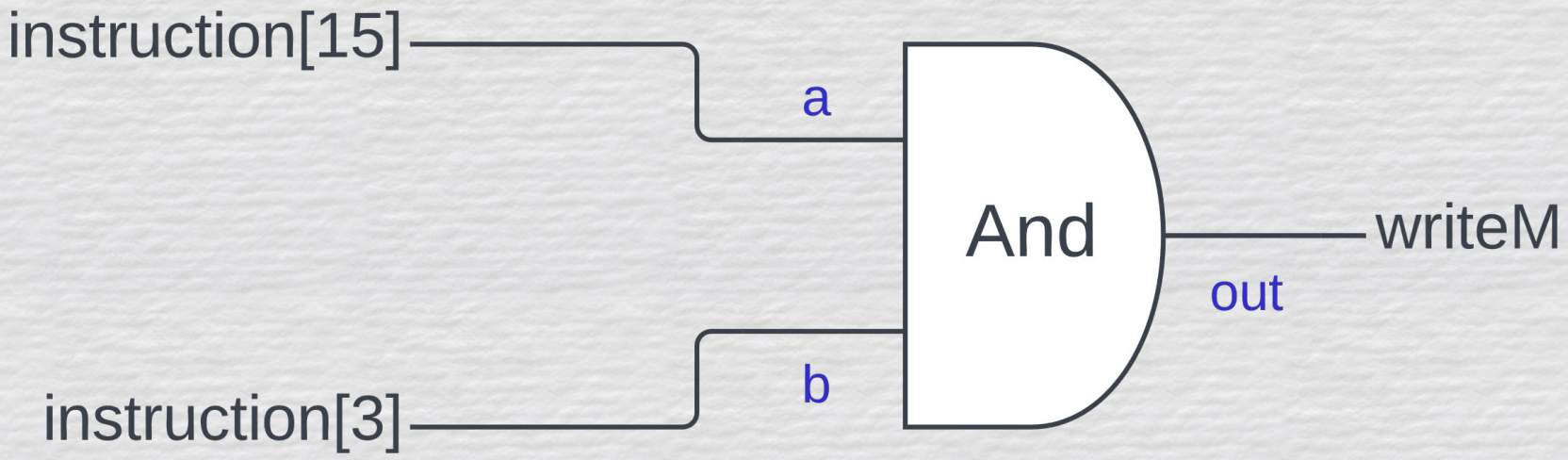
D-Register


instruction[15]
a
And
instruction[4]
b
out
loadD
load
DRegister
out
outD
16
in
outALU
16

D-Register – Code

```
85      //=============================================================================================
86      // D-Register
87      // When instruction[4] == 1, it is D=outALU which means D should load a value
88      And(a=, b=, out=);
89
90      /*
91          Loads a new input to the D-Register if above code dictates it to
92          Set the ALU x-input to D
93      */
94      DRegister(in=, load=, out=);
```

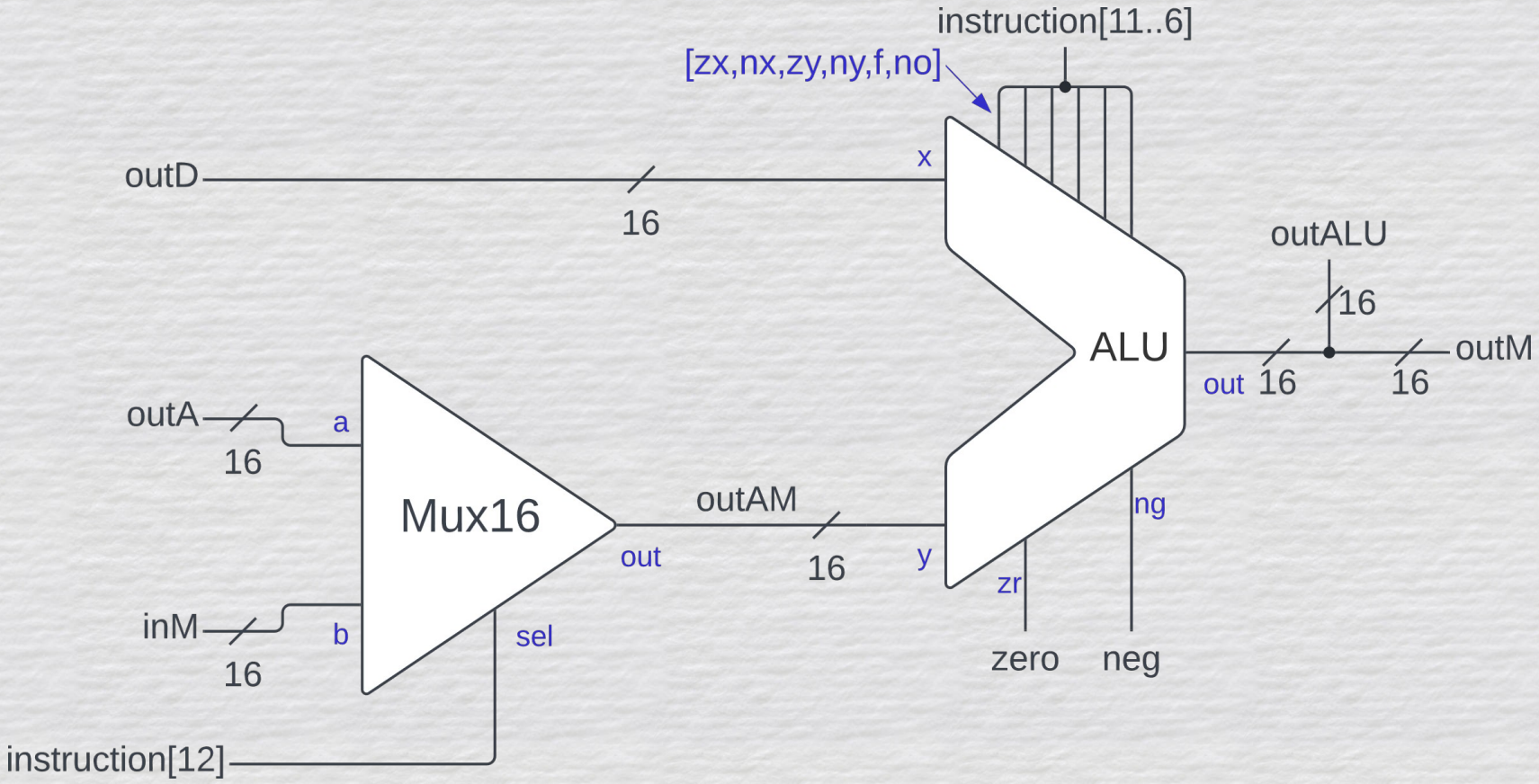
M-Register



M-Register – Code

```
96      //=============================================================================================
97      // M-Register
98      // When instruction[3] == 1, it is M=outALU which means M should load a value
99      And(a=, b=, out=);
```

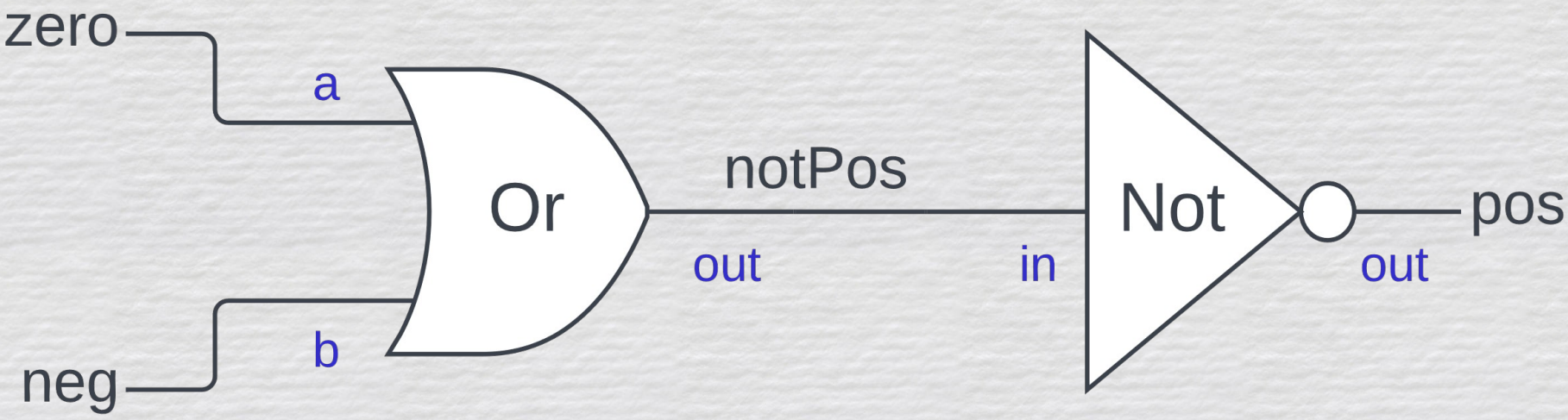
ALU


instruction[11..6]
[zx,nx,zy,ny,f,no]
outD
16
x
outALU
16
ALU
outM
16
out
16
16
outA
16
a
Mux16
outAM
16
out
y
ng
zr
zero
neg
inM
16
b
sel
instruction[12]

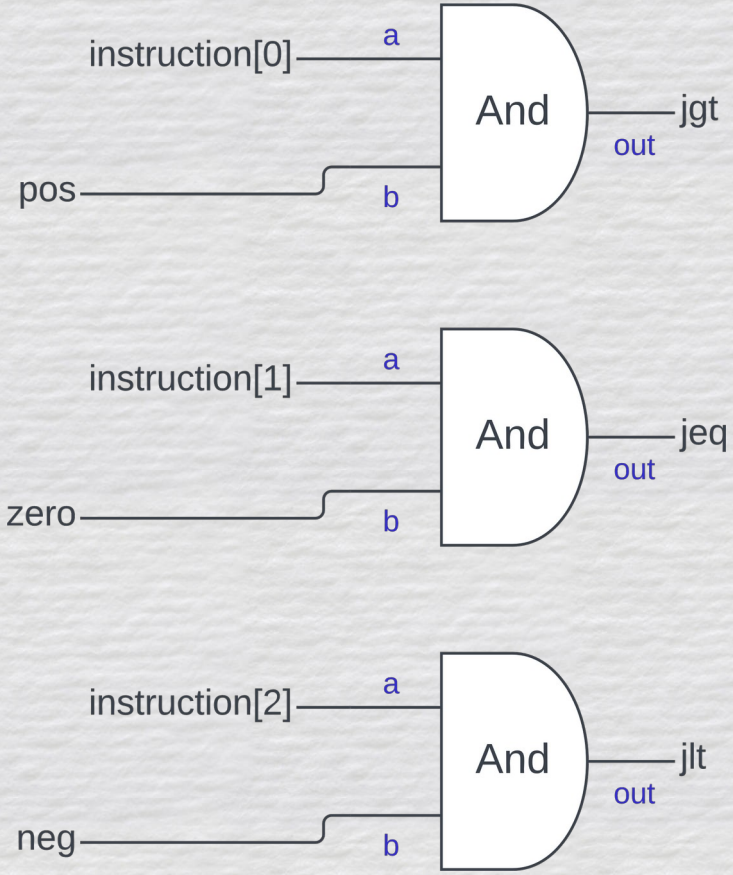
ALU – Code

```
101     //=============================================================================================================
102     // ALU
103     // Set the ALU y-input to either A or M (RAM[A])
104     Mux16(a=, b=, sel=, out=);
105
106     // Set the ALU appropriately
107     ALU(x=, y=,          // Set the ALU inputs
108         zx=, nx=,        // Set the X Pins
109         zy=, ny=,        // Set the Y Pins
110         f=, no=,         // Set the Function Pins
111         out=, out=,      // Set the ALU Outputs
112         zr=, ng=);       // Set the Output Flags
```

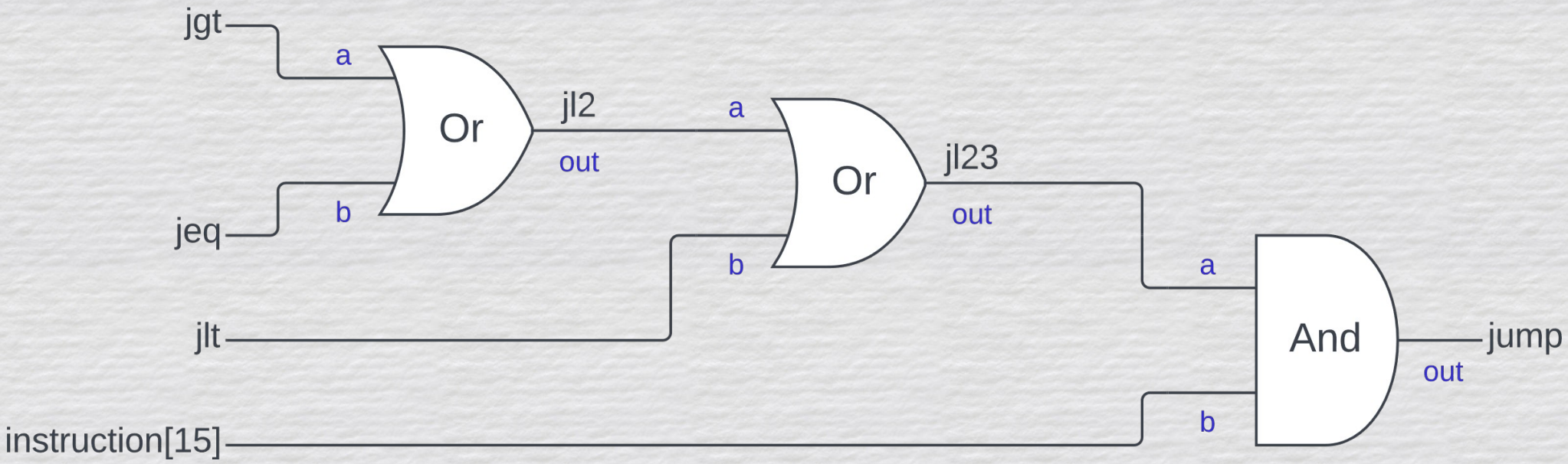
Jump – Pt. 1



Jump – Pt. 2


instruction[0]
a
And
jgt
out
pos
b
instruction[1]
a
And
jeq
out
zero
b
instruction[2]
a
And
jlt
out
neg
b

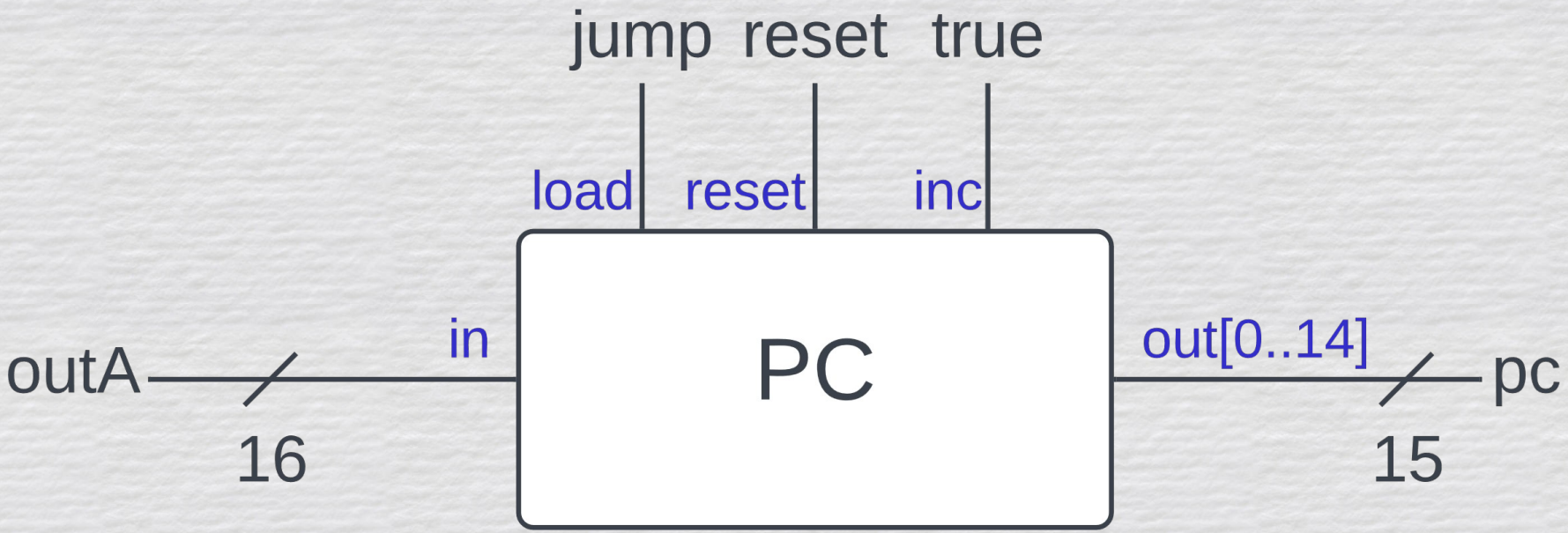
Jump – Pt. 3


jgt
a
Or
jl2
out
a
Or
jl23
out
jeq
b
b
a
jlt
And
jump
out
b
instruction[15]

Jump – Code

```
114     //=============================================================================================================
115     // Jump
116     // zero | neg ALU flags to determine if neither is true
117     Or(a=, b=, out=);
118     // Negate the result to determine if positive is true
119     Not(in=, out=);
120
121     And(a=, b=, out=);      // outALU > 0
122     And(a=, b=, out=);     // outALU == 0
123     And(a=, b=, out=);      // outALU < 0
124
125     // Combine the first 2 jump bits
126     Or(a=, b=, out=);
127     // Combine the prior jump bits and 3rd jump bit
128     Or(a=, b=, out=);
129
130     // When j123 == 1, the jump condition is met
131     And(a=, b=, out=);
```

PC Register



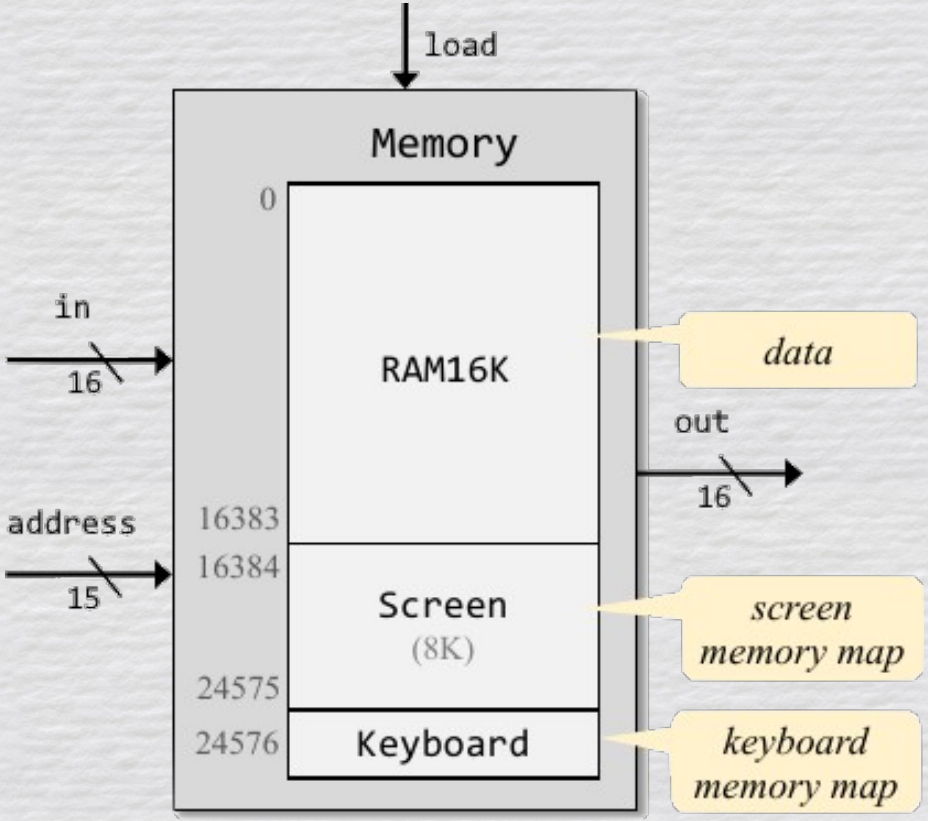
PC Register – Code

```
133     //=============================================================================================
134     // PC Register
135     /*
136         Takes in the A-Register, jump result, reset input, and increment (always true)
137         to determine where the next address in ROM32K should be
138
139         if (jump == 1)
140             pc = outA (we jump to ROM[A])
141         else if (reset == 1)
142             pc = 0 (we jump to ROM[0])
143         else
144             pc = pc + 1 (we increment to our next address)
145     */
146     PC(in=, load=, reset=, inc=, out[0..14]=);
```

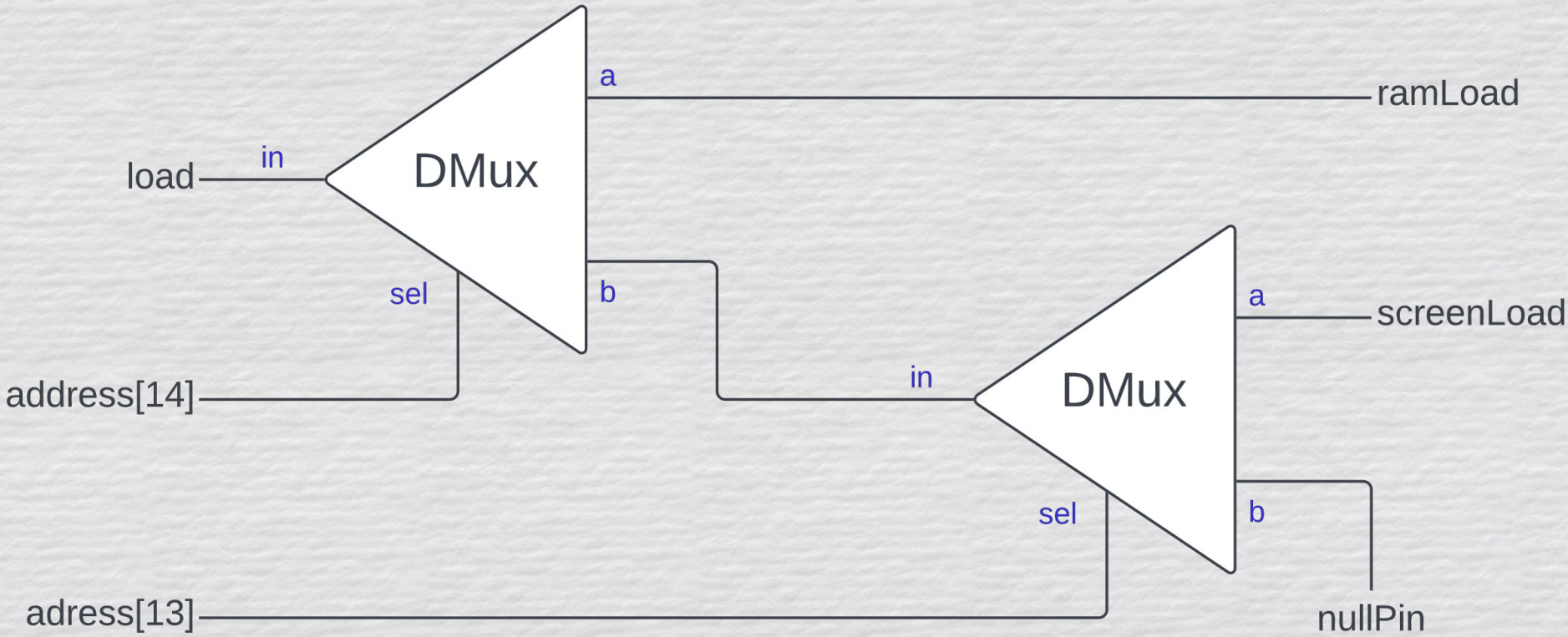
Memory

- The Memory chip is far easier to implement than our CPU
- It simply consists of 3 main components
  - RAM16K
  - Screen
  - Keyboard
- We simply need to use DMuxes and Mux16s to select which of these we would like to use

Memory – Overview



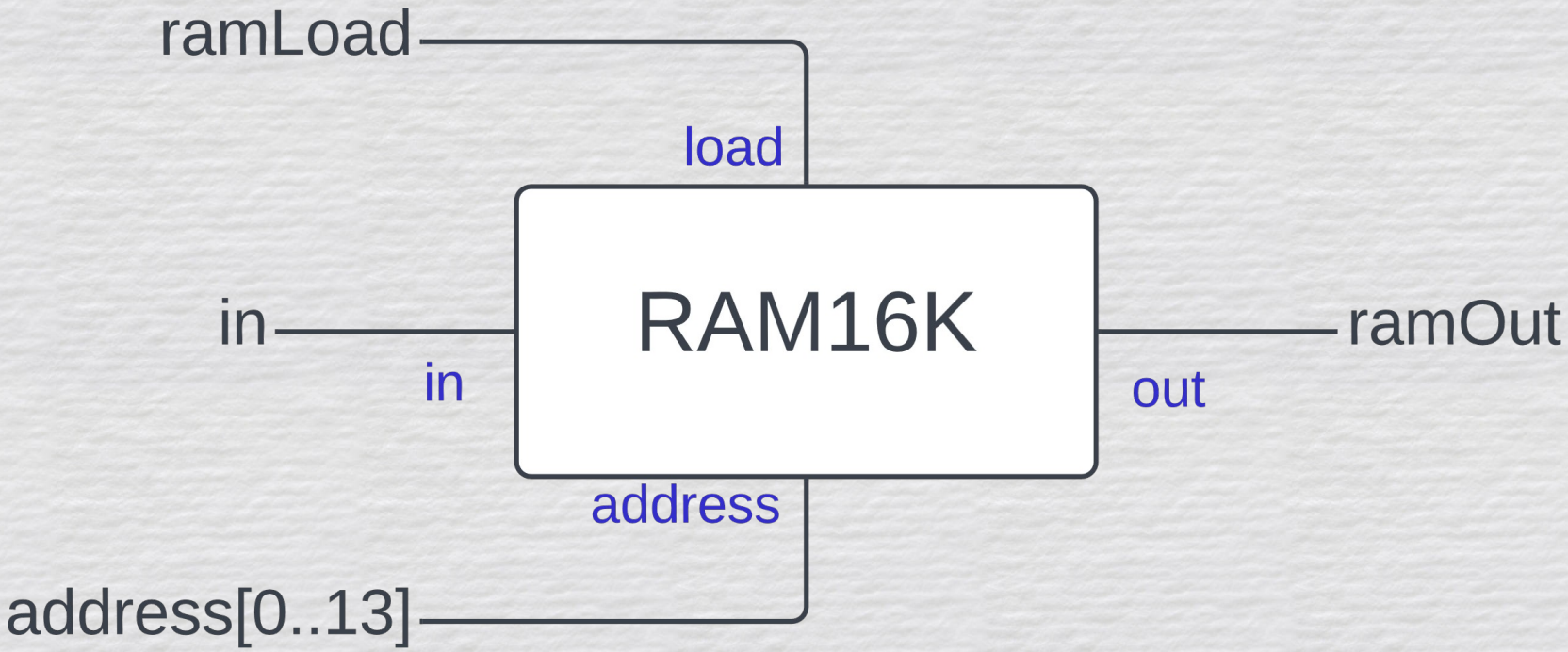
Determine Memory Input



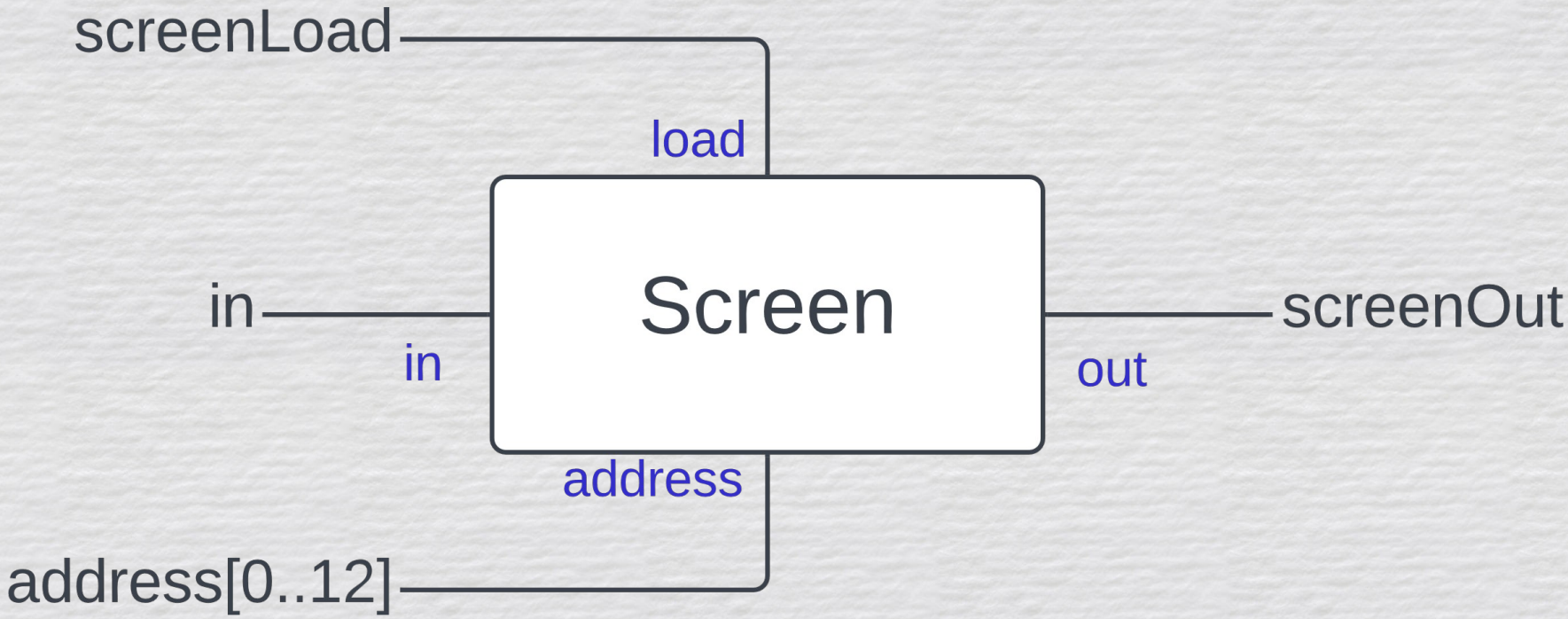
Determine Memory Input – Code

```
30      //=============================================================================================================
31      // Determine Input (DMux)
32      /*
33          if (address[14] == 0)
34              We are using RAM (ramLoad)
35          else
36              We are using either Screen or Keyboard (skLoad)
37      */
38      DMux(in= ,sel= ,a= ,b= );
39      /*
40          if (address[14] == 1)
41          {
42              if (address[13] == 0)
43                  We are using Screen (screenLoad)
44              else
45                  We are using Keyboard and don't use the load input (nullPin)
46          }
47      */
48      DMux(in= ,sel= ,a= ,b= );
```

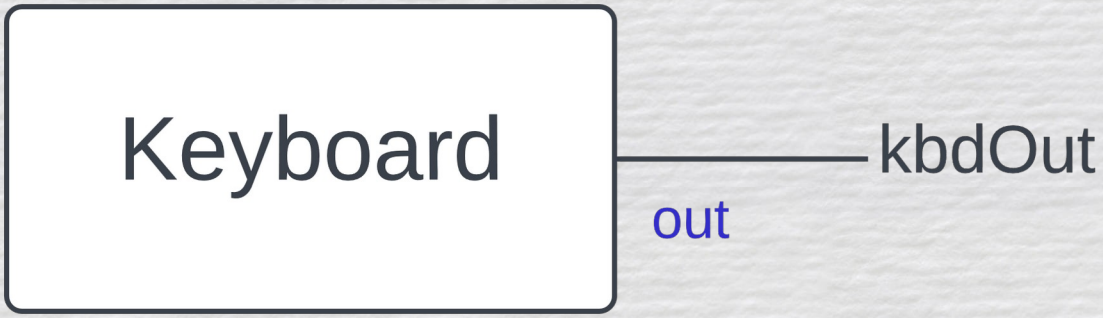
Handle Components – RAM 16K



Handle Components – Screen



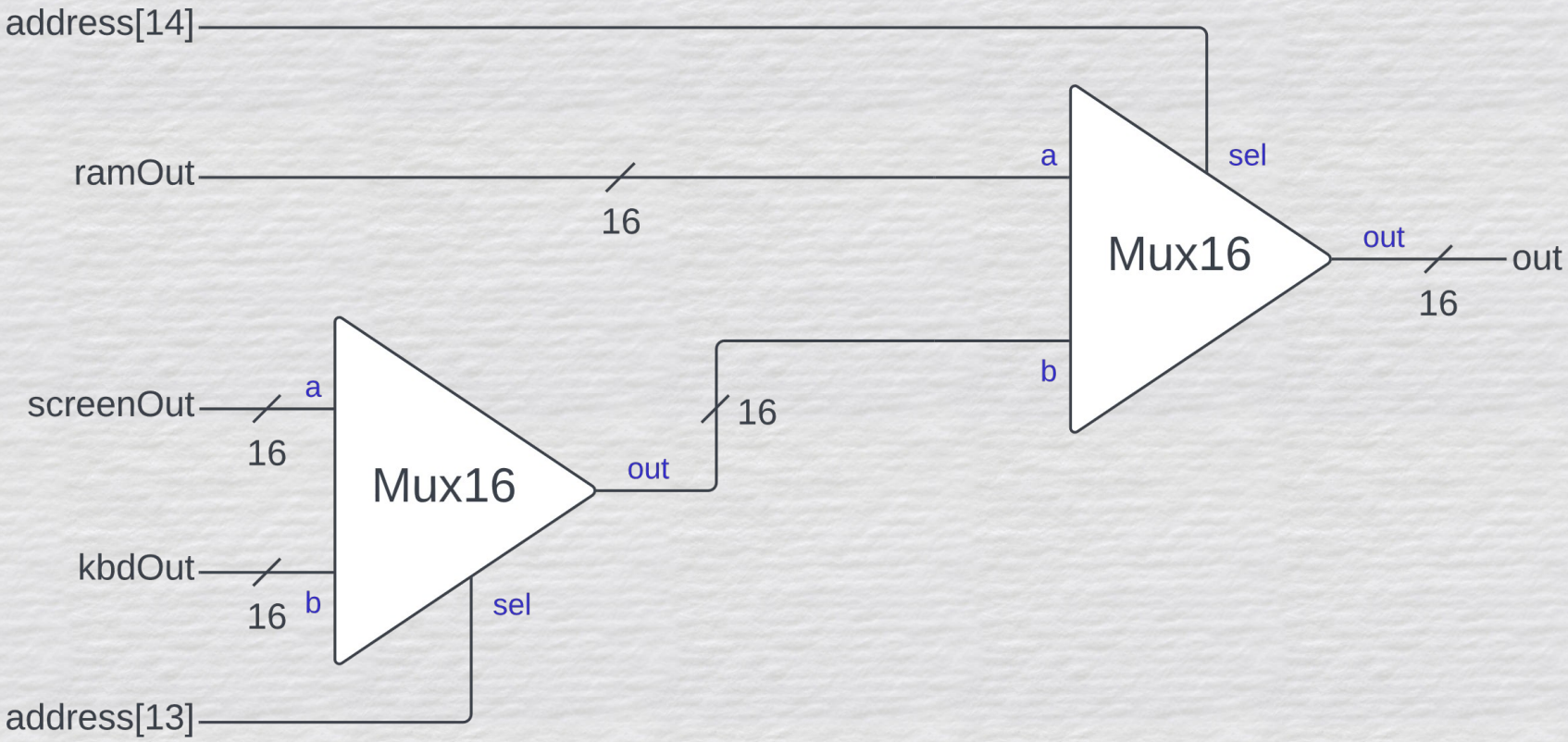
Handle Components – Keyboard



Handle Components – Code

```
47      //=============================================================================================
48      // Handle the outputs of the RAM/Screen/Keyboard
49      // RAM16K using the aformentioned ramLoad and the first 14 bits of the address (ramOut)
50      RAM16K(in=, load=, address=, out=);
51      // Screen (RAM8K) using the aformentioned screen and the first 13 bits of the address (screenOut)
52      Screen(in=, load=, address=, out=);
53      // Simple Keyboard (kbdOut)
54      Keyboard(out=);
```

Determine Memory Output



Determine Memory Output – Code

```
59  //=============================================================================================
60   // Determine Output (Mux16)
61   /*
62       if (address[13] == 0)
63           We are using Screen (screenOut)
64       else
65           We are using Keyboard (kbdOut)
66   */
67   Mux16(a= ,b= ,sel= ,out= );
68   /*
69       if (address[14] == 0)
70           We are using RAM (ramOut)
71       else
72           We are using either Screen or Keyboard (skLoad)
73   */
74   Mux16(a= ,b= ,sel= ,out= );
```