We acknowledge and pay our respects to the Kaurna people,
the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

THE UNIVERSITY *of* ADELAIDE

Computer Systems

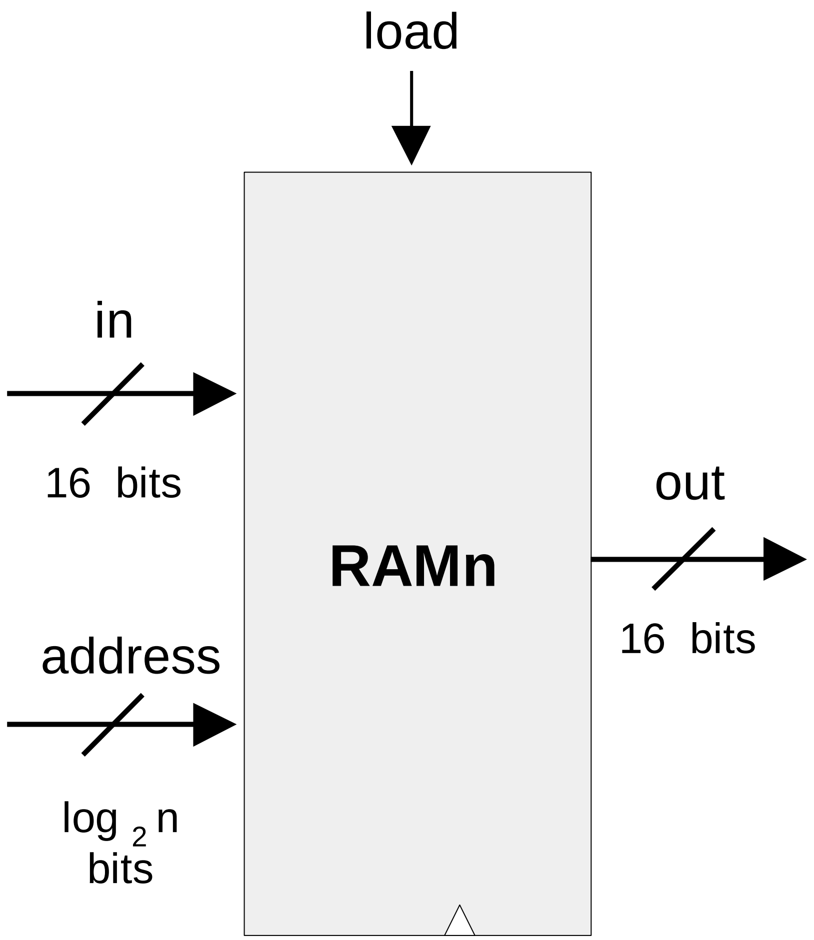
Lecture 04: System Architecture and Machine Language

Review: The whole system

- **Programming Language**
  - Compiler
  - Chapters 10-11
- **Virtual Machine**
  - Translator
  - Chapters 7-8
- **Assembly Language**
  - Assembler
  - Chapters 6
- **Machine Language**
  - System Architecture
  - Chapters 4-5
- **Hardware Platform**
  - Gate Logic
  - Chapters 1-3
- **Chips and Logic Gates**
  - Electronics

THE UNIVERSITY of ADELAIDE

Review: RAM interface

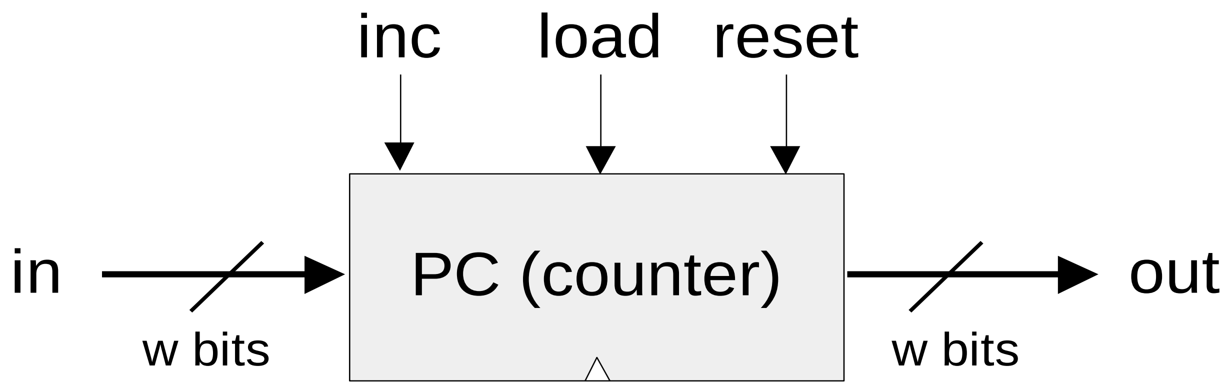



```
Chip name:  RAMn  // n and k are listed below
Inputs:     in[16], address[k], load
Outputs:    out[16]
Function:   out(t)=RAM[address(t)](t)
            If load(t-1) then
               RAM[address(t-1)](t)=in(t-1)
Comment:    "=" is a 16-bit operation.
```

The specific RAM chips needed for the Hack platform are:

Chip name	n	K
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

Review: Program Counter - Diagram



- When reset pin 1, counter resets (outputs 0 next clock tick)
- Else when load pin 1, counter set to input (outputs in next clock tick)
- Else when inc pin 1, counter increments (outputs current value + 1 next tick)
- Else counter unchanged (outputs current value next tick)

Review: Instructions for a Basic Computer

- Mathematical operations
- Logical operations
- Flow of control instructions
- Conditions
- Write to memory
- Reading from memory
- Do nothing
- …

Review: Machine language

- Machine language typically refers to the binary representation of a program.
- Assembly language is a human readable description of machine language, as on the previous slide.
- Machine language usually provides a way to embed explicit values in a program, eg the ADDI instruction.
- Machine language is usually specific to the hardware and / or assembler program that translates assembly language into binary.

THE UNIVERSITY of ADELAIDE

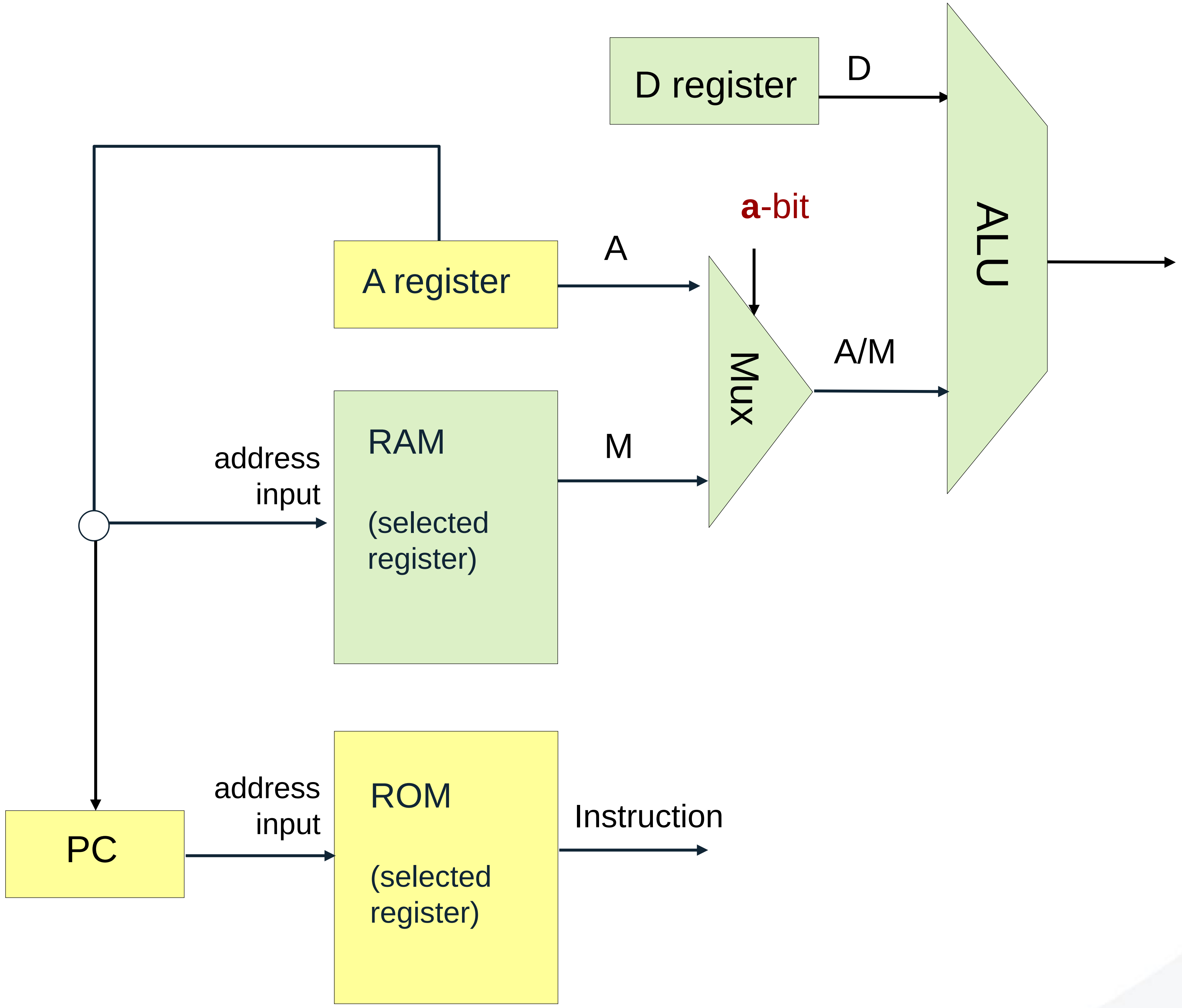
Review: The Hack computer

A 16-bit machine consisting of the following elements:

- Data memory: **`RAM`** – an addressable sequence of registers
- Instruction memory: **`ROM`** – an addressable sequence of registers
- Registers: **`D`**, **`A`**, **`M`**, where **`M`** stands for **`RAM[A]`**
- Processing: **`ALU`**, capable of computing various functions
- Program counter: **`PC`**, holding an address
- Control: The **`ROM`** is loaded with a sequence of 16-bit instructions, one per memory location, beginning at address 0. Fetch-execute cycle: later
- Instruction set: Two instructions types: A-instruction, C-instruction.

System Architecture

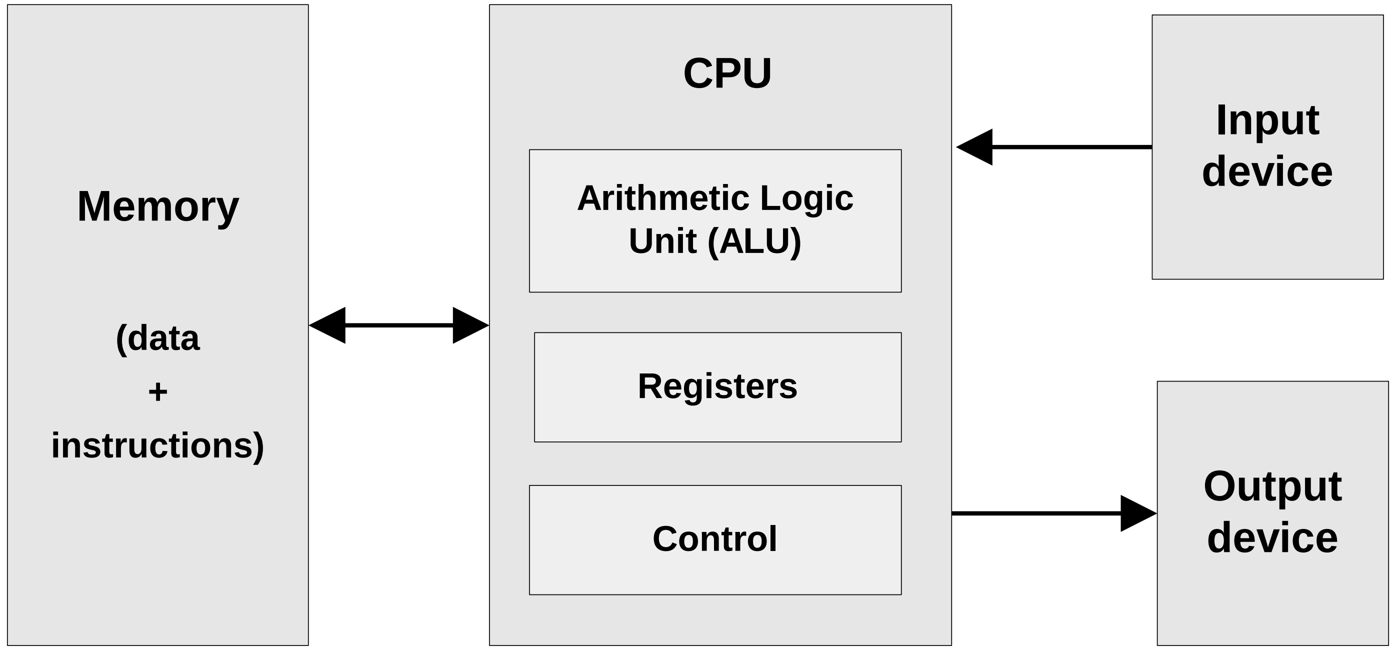
A basic architecture

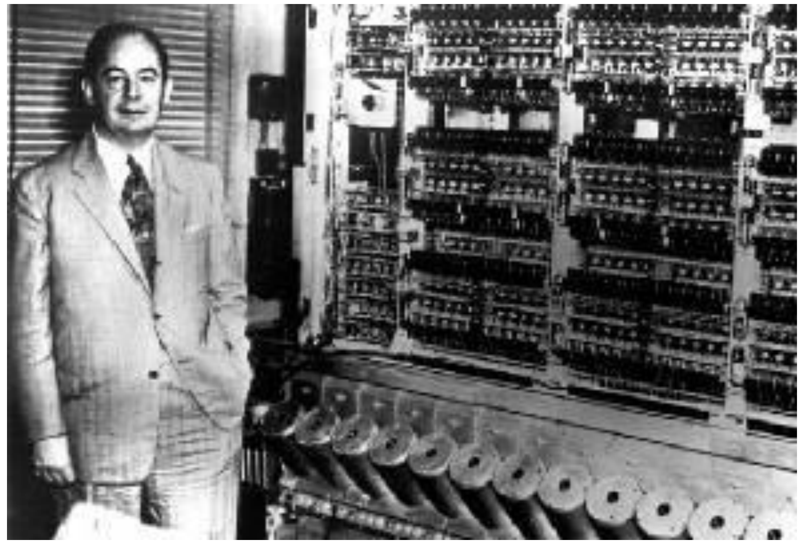


Early Computers

- **We have been using computers for a very very long time but not in the form we are using today.**
  - Early machines were used to predict astronomical positions for calendars and astrology.
- **The Antikythera mechanism, possibly 100 to 200 BC**
- its sophistication suggests such devices existed much earlier
- **The first programmable analogue computer?**
  - Al-Jazari built a programmable water powered clock c. 1206
- **Slide rules**
  - invented around 1620 with the discovery of logarithms
  - heavily used by nearly everyone until the 1970s

Von Neumann machine (circa 1940)





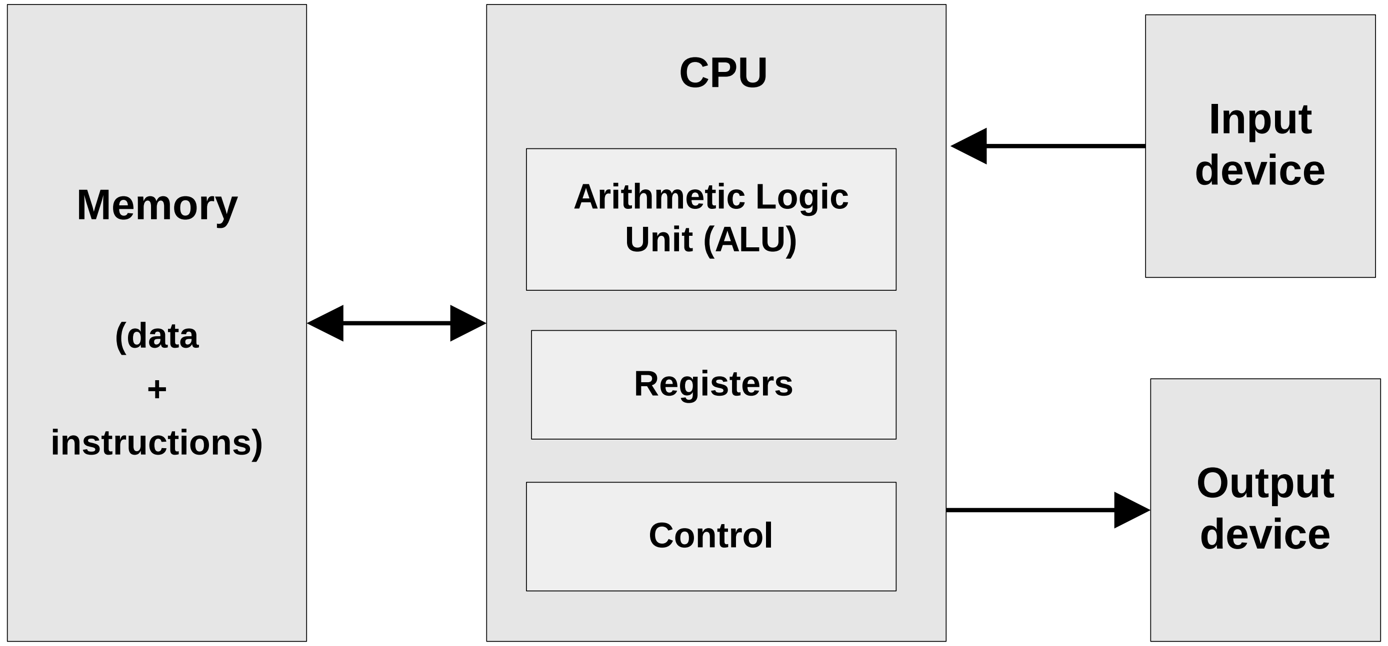
Stored program concept!

John Von Neumann (and others) ... made it possible

Andy Grove (and others) ... made it small and fast.

THE UNIVERSITY of ADELAIDE

Processing logic: fetch-execute cycle



Executing the *current instruction* involves one or more of the following micro-tasks:

- Have the ALU compute some function $\text{out} = f\,(\text{register values})$
- Write the ALU output to selected registers
- As a side-effect of this computation,
  figure out which instruction to fetch and execute next.

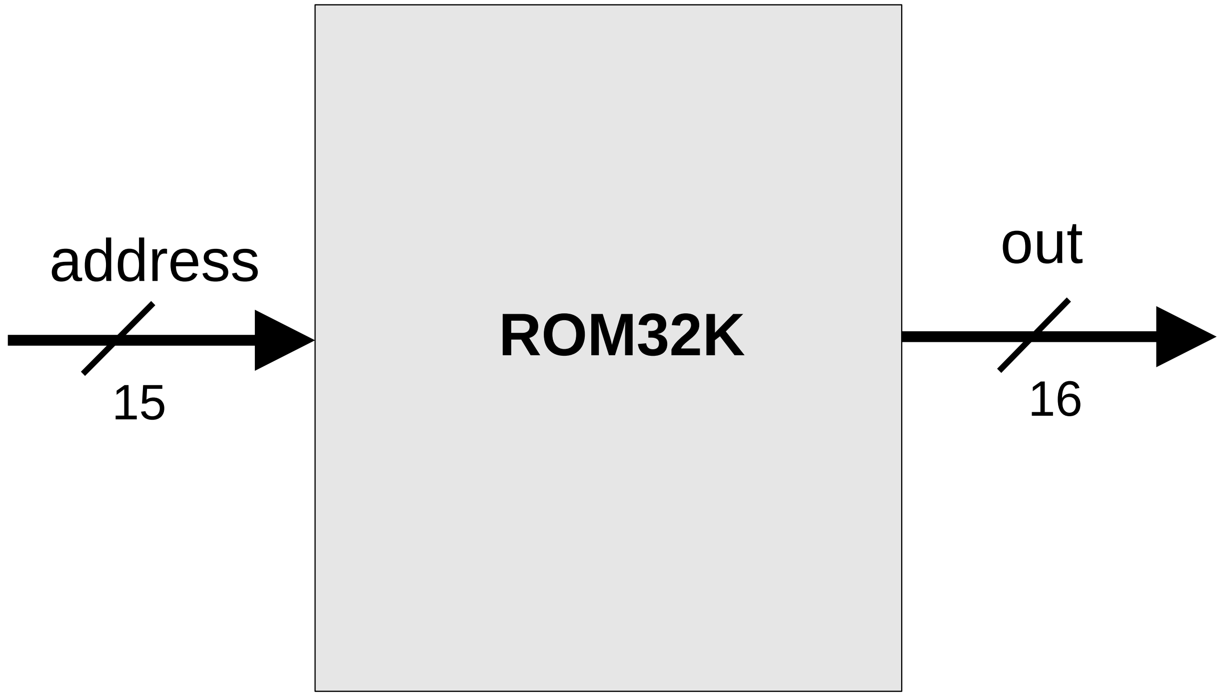
The Hack computer

- A 16-bit Von Neumann platform
- The *instruction memory* and the *data memory* are physically separate
- Screen: 512 rows by 256 columns, black and white
- Keyboard: standard
- Designed to execute programs written in the Hack machine language
- Can be easily built from the chip-set that we built so far in the course

Main parts of the Hack computer:

- Instruction memory (ROM)
- Memory (RAM):
    - Data memory
    - Screen (memory map)
    - Keyboard (memory map)
- CPU
- Computer (**the logic that holds everything together).**

Instruction memory



<u>Function:</u>

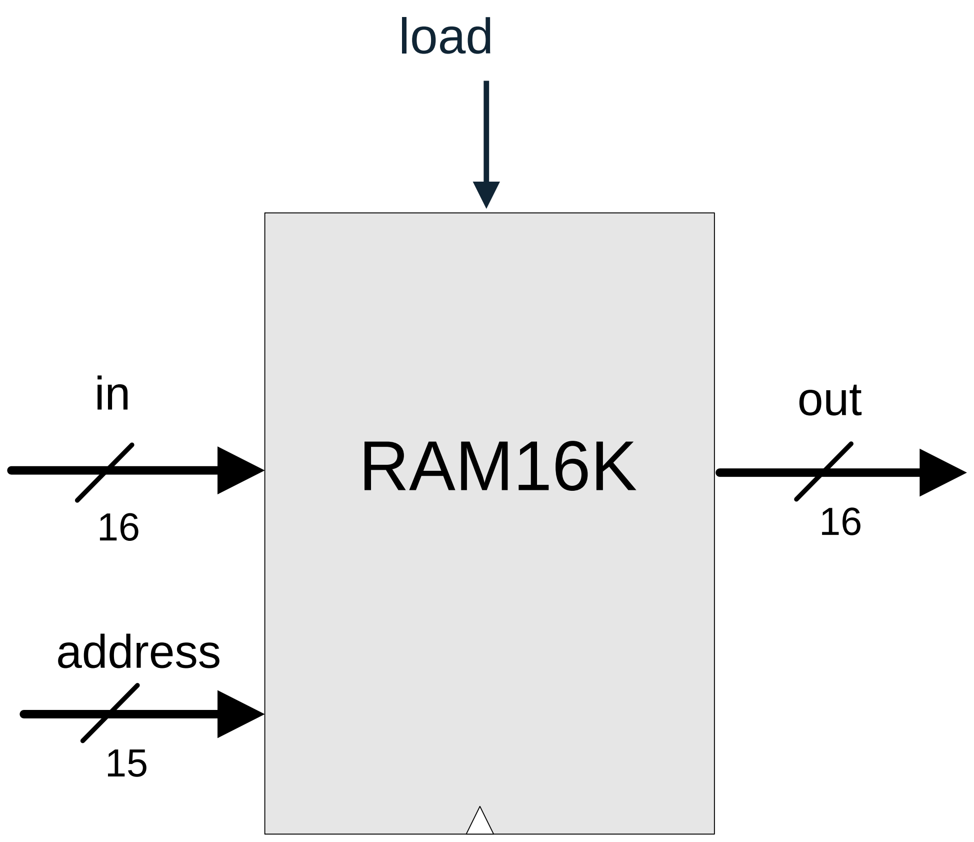
The `ROM` is pre-loaded with a program written in the Hack machine language

The `ROM` chip always emits a 16-bit number:

```
out = ROM32K[address]
```

This number is interpreted as the *current instruction*.

Data memory



<u>Low-level (hardware) read/write logic:</u>

To read `RAM[k]`: set `address` to k,
probe `out`

To write `RAM[k]=x`: set `address` to k,
set `in` to x,
set `load` to 1,
run the clock

<u>High-level (OS) read/write logic:</u>

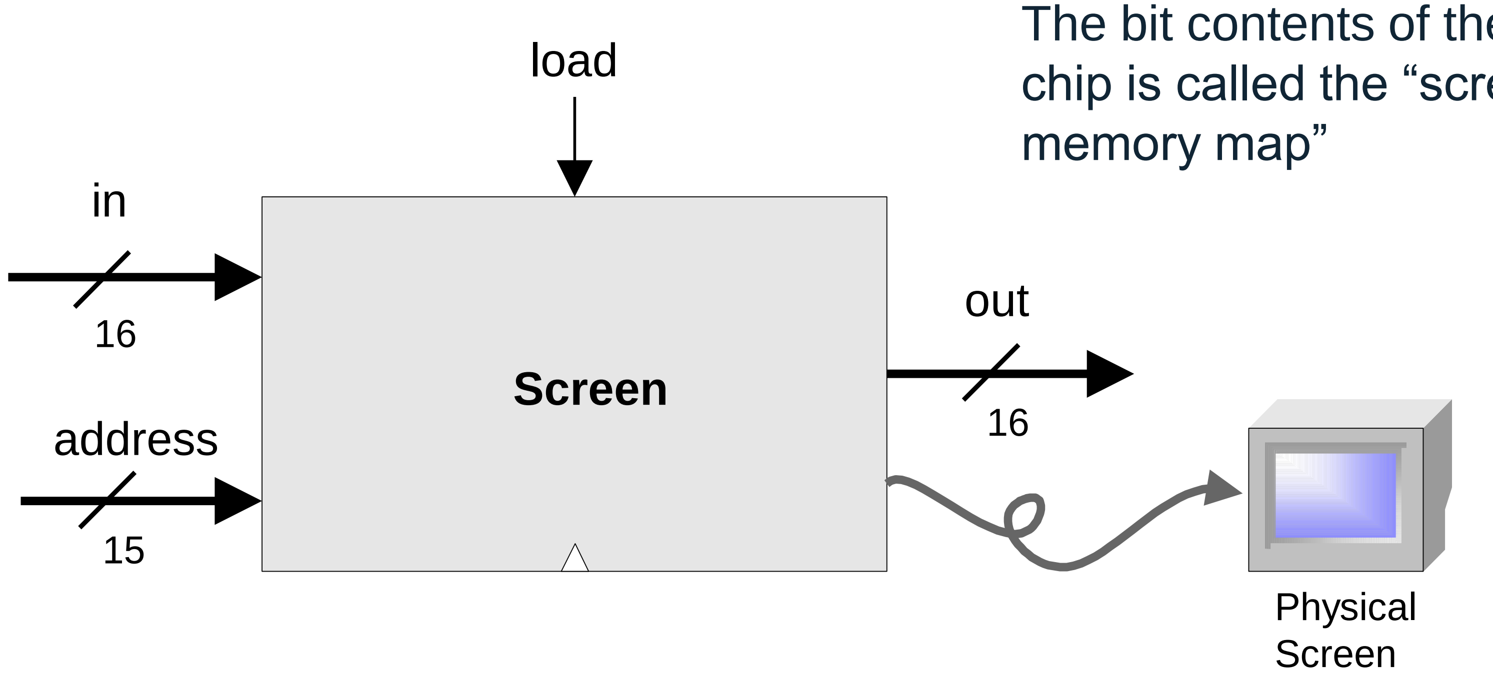
To read `RAM[k]`: use the OS command `out = peek(k)`

To write `RAM[k]=x`: use the OS command `poke(k,x)`

`peek` and `poke` are OS commands whose implementation should effect the same behavior as the low-level commands

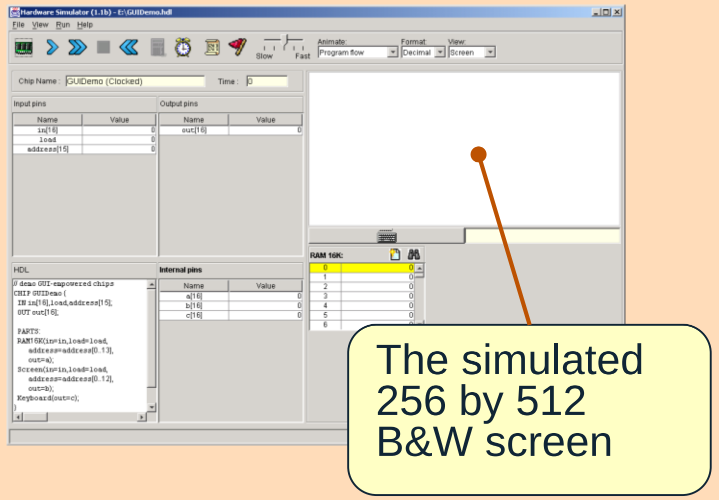
More about `peek` and `poke` this later in the course, when we'll write the OS.

Screen



The bit contents of the Screen chip is called the “screen memory map”

Simulated screen:



When loaded into the hardware simulator, the built-in Screen.hdl chip opens up a screen window; the simulator then refreshes this window from the screen memory map several times each second.

The Screen chip has a basic RAM chip functionality:

- read logic: `out = Screen[address]`
- write logic: `if load then Screen[address] = in`

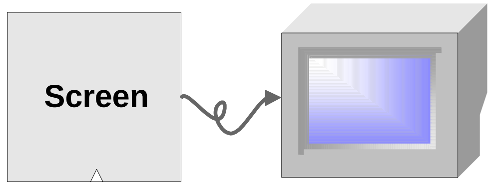
<u>Side effect:</u>

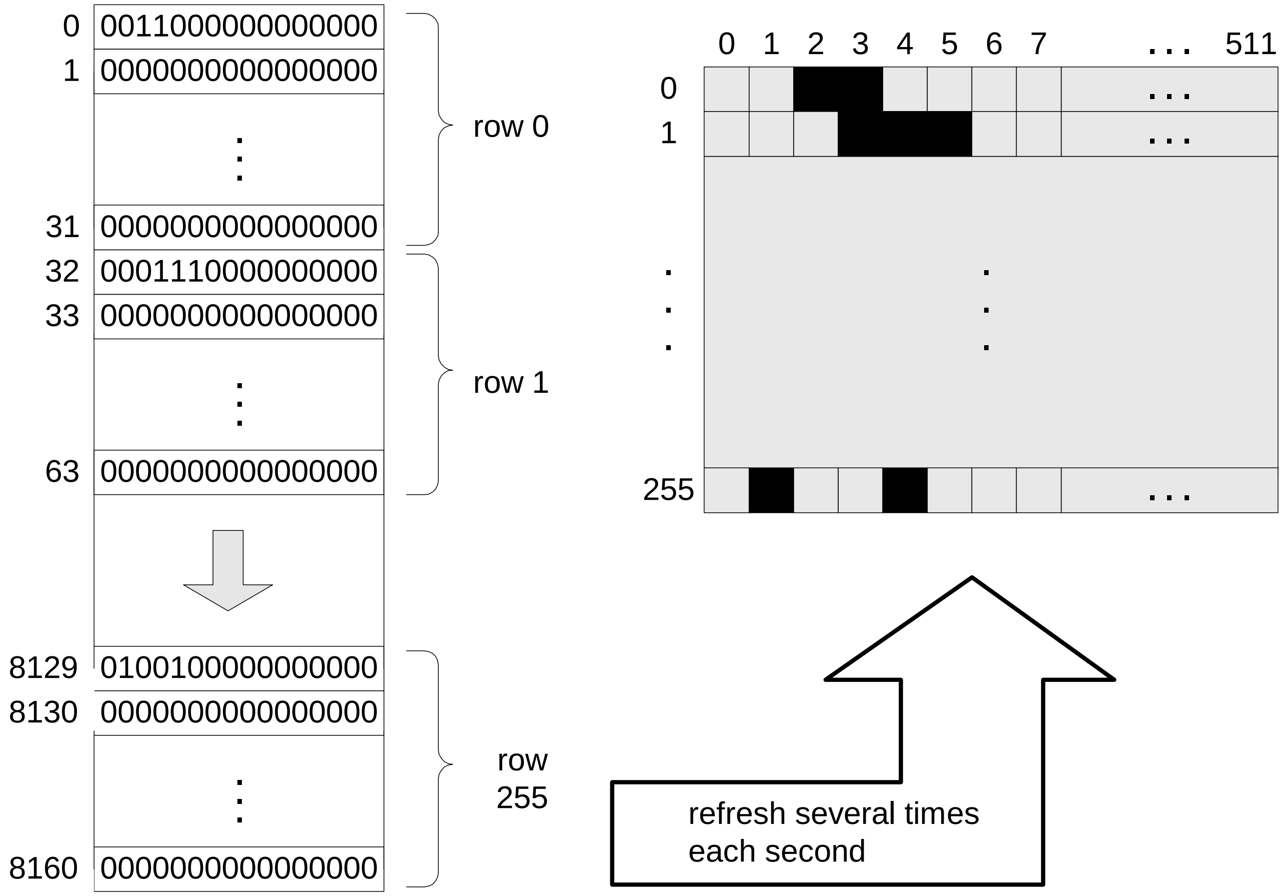
Continuously refreshes a 256 by 512 black-and-white screen device

THE UNIVERSITY of ADELAIDE

Screen memory map

In the Hack platform, the screen is implemented as an 8K 16-bit RAM chip.





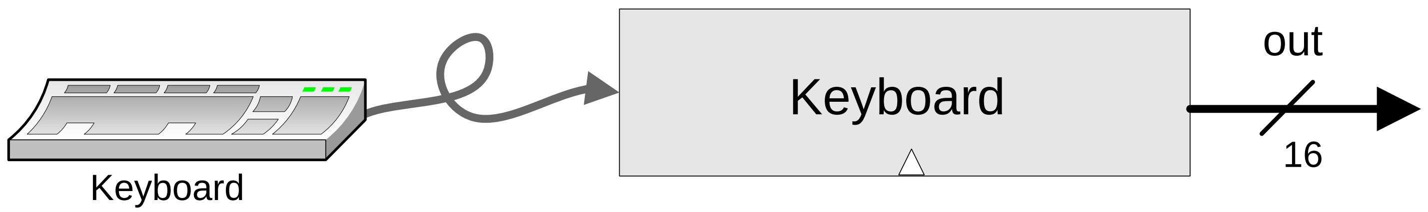
How to set the `(row,col)` pixel of the screen to black or to white:

- **Low-level (machine language): Set the `col%16` bit of the word found at `Screen[row*32+col/16]` to `1` or to `0`**
  (`col/16` is integer division)

- **High-level: Use the OS command `drawPixel(row,col)`**

  (effects the same operation, discussed later in the course, when we'll write the OS).

Keyboard

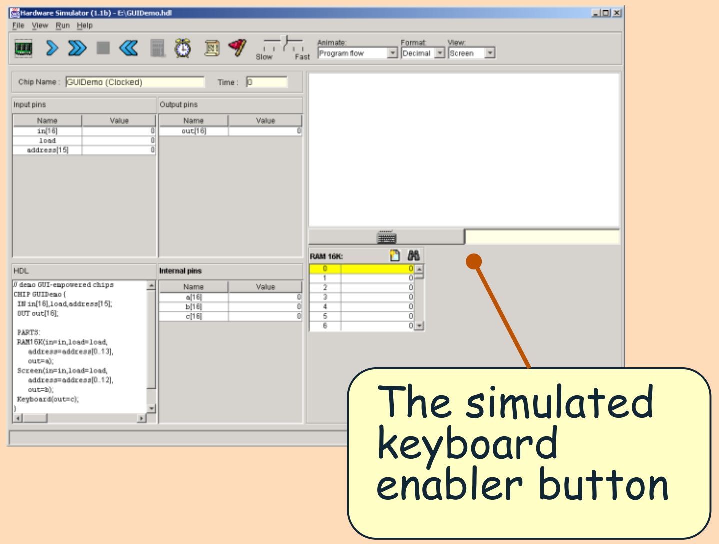


Keyboard chip: a single 16-bit register

Input: scan-code (16-bit value) of the currently pressed key, or 0 if no key is pressed

Output: same

Simulated keyboard:



The keyboard is implemented as a built-in **Keyboard.hdl** chip. When this java chip is loaded into the simulator, it connects to the regular keyboard and pipes the scan-code of the currently pressed key to the keyboard memory map.

Special keys:

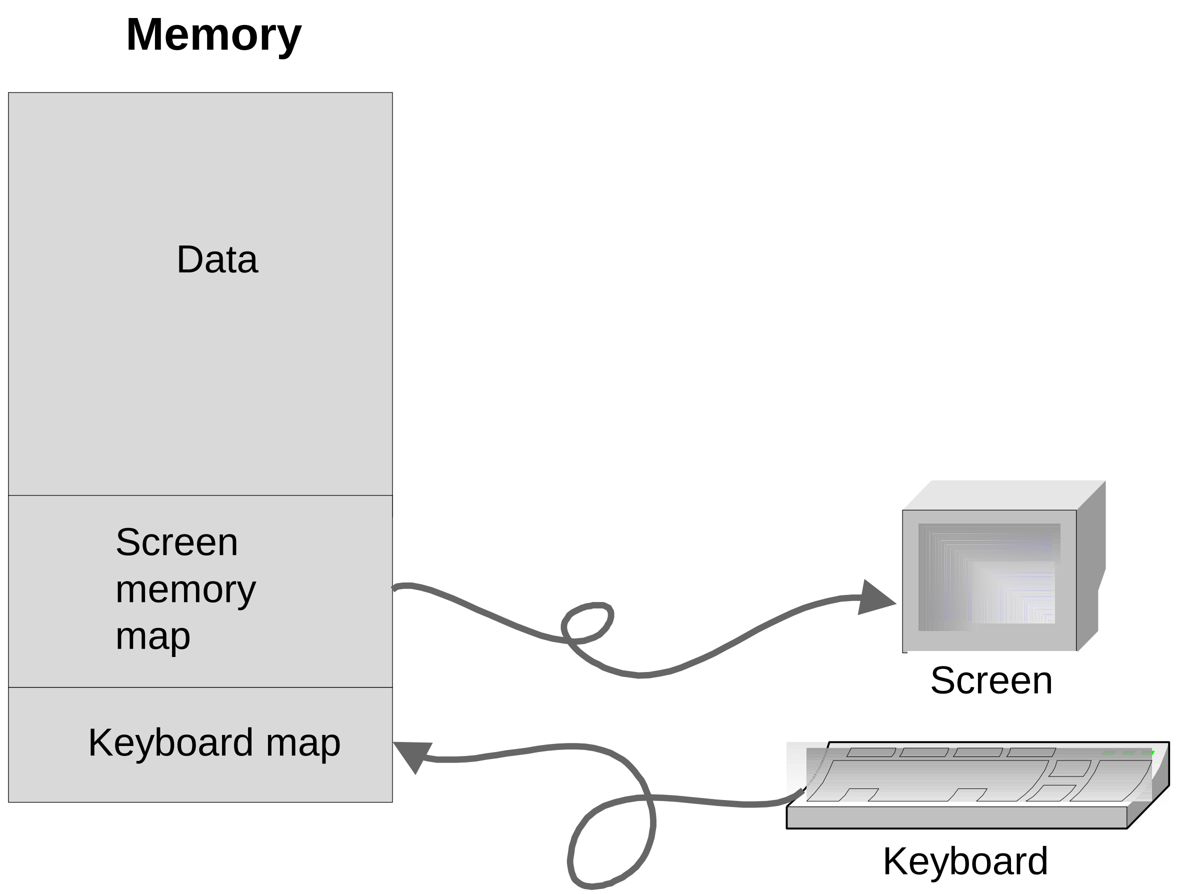
Key pressed	Keyboard output
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134

Key pressed	Keyboard output
end	135
page up	136
page down	137
insert	138
delete	139
esc	140
f1-f12	141-152

How to read the keyboard:

- Low-level (hardware): probe the contents of the Keyboard chip
- High-level: use the OS command keyPressed()
  (effects the same operation)

Memory Mapping – Conceptual view

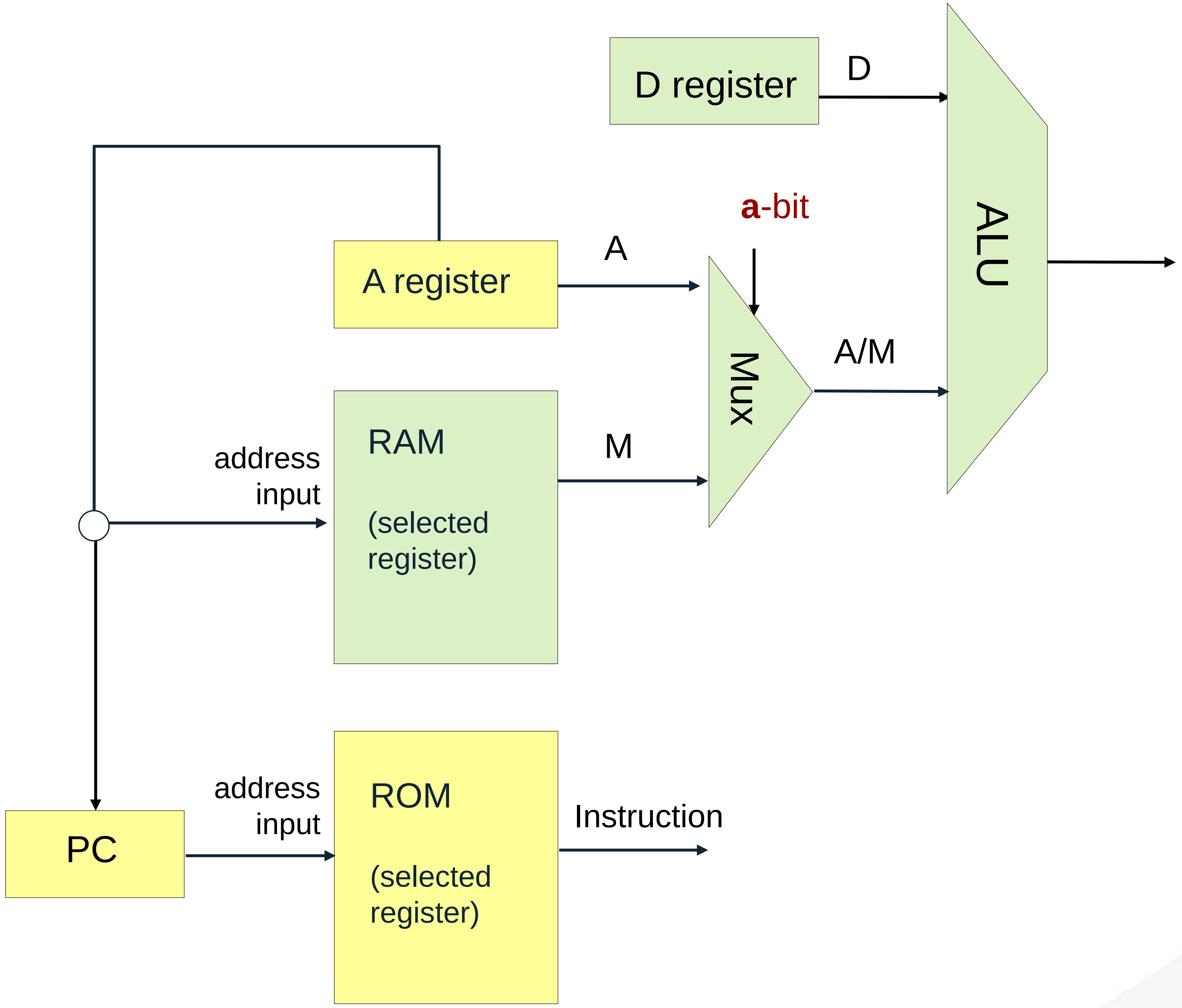


From the program's point of view our RAM is just 32K words of read/write memory

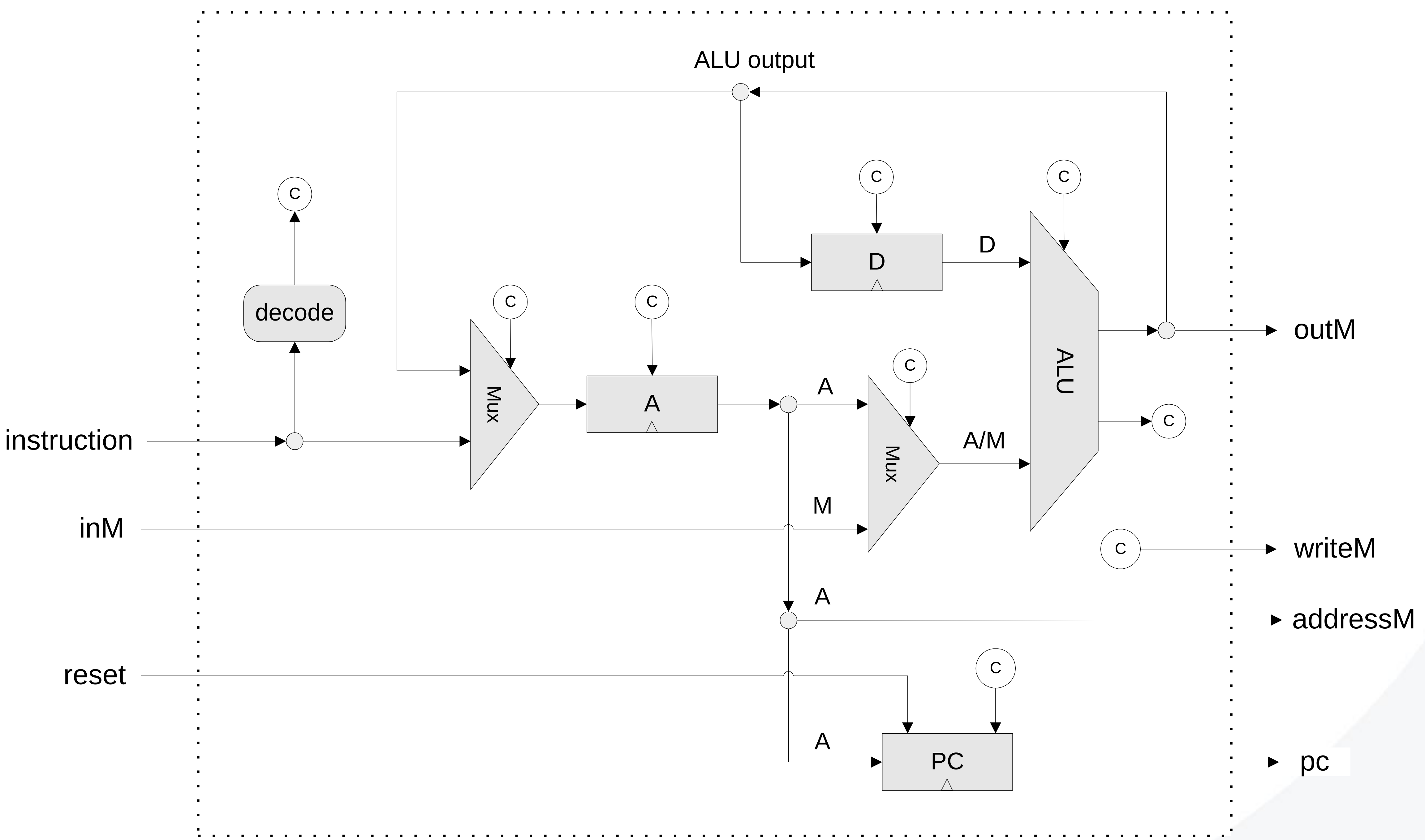
<u>Using the memory:</u>

- To record or recall values (e.g. variables, objects, arrays), use the first 16K words of the memory
- To write to the screen (or read the screen), use the next 8K words of the memory
- To read which key is currently pressed, use the next word of the memory.

A simplified architecture



Looking Closer



The **C**-instruction

dest = comp; jump

	comp									dest			jump		
binary: 1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
i15	i14	i13	i12	i11	i10	i9	i8	i7	i6	i5	i4	i3	i2	i1	i0

(when a=0) *comp*	c1	c2	c3	c4	c5	c6	(when a=1) *comp*
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

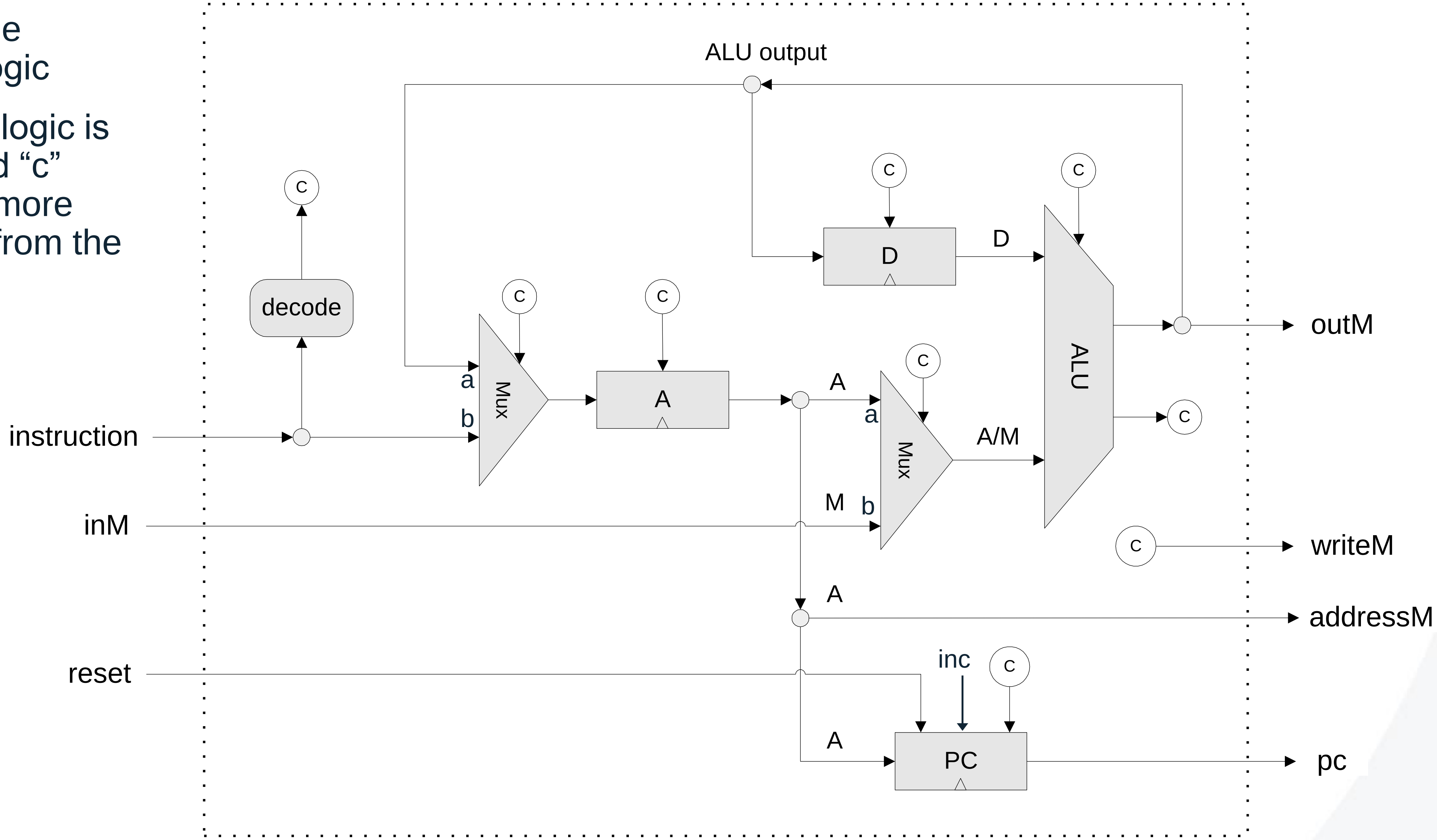
j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	**Mnemonic**	**Effect**
0	0	0	null	No jump
0	0	1	JGT	If *out* > 0 jump
0	1	0	JEQ	If *out* $= 0$ jump
0	1	1	JGE	If *out* ≥ 0 jump
1	0	0	JLT	If *out* < 0 jump
1	0	1	JNE	If *out* $\neq 0$ jump
1	1	0	JLE	If *out* ≤ 0 jump
1	1	1	JMP	Jump

CPU implementation

dest = comp; jump				*comp*						*dest*			*jump*		
binary: 1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
i15	i14	i13	i12	i11	i10	i9	i8	i7	i6	i5	i4	i3	i2	i1	i0

Chip diagram:

- Includes most of the CPU's execution logic
- The CPU's control logic is hinted: each circled "c" represents one or more control bits, taken from the instruction
- The "decode" bar does not represent a chip, but rather indicates that the instruction bits are decoded somehow.



Cycle:

- Execute
- Fetch

Execute logic:

- Decode
- Execute

Fetch logic:

If there should be a jump,
set PC to A
else set PC to PC+1

Resetting the computer:

Set reset to 1,
then set it to 0.

Observations

We can use individual bits from the C-instruction to control:

The ALU (multiple wires)

The Choice of Source (A,D,M)

The Choice of Destination (A,D,M)

Some Input signals to the PC to let it know if it should be updated with the value of A

- Note that this decision also depends on some output status bits from the ALU..

Machine Language

The A-instruction

```
@value        // A ← value
```

Where *value* is either a number or a symbol referring to some number.

Used for:

Coding example:

Entering a constant value

(`A = value`)

```
@17     // A = 17
D = A   // D = 17
```

- Selecting a **RAM** location
  (`register = RAM[A]`)

```
@17     // A = 17
D = M   // D = RAM[17]
```

- Selecting a **ROM** location
  (`PC = A`)

```
@17     // A = 17
JMP     // fetch the instruction
        // stored in ROM[17]
```

THE UNIVERSITY of ADELAIDE

The C-instruction

$dest = x + y$

$dest = x - y$

$dest = x$

$dest = 0$

$dest = 1$

$dest = -1$

x = {A, D, M}

y = {A, D, M , 1}

$dest$ = {A, D, M, MD, A, AM, AD, AMD, null}

Exercise: In small groups implement the following tasks using Hack :

- Set **D** to **A-1**
- Set both **A** and **D** to **A + 1**
- Set **D** to **19**
- Set both **A** and **D** to **A + D**
- Set **RAM[5034]** to **D - 1**
- Set **RAM[53]** to **171**
- Add **1** to **RAM[7]**,
  and store the result in **D.**

The C-instruction

$dest = x + y$

$dest = x - y$

$dest = x$

$dest = 0$

$dest = 1$

$dest = -1$

x = {A, D, M}

y = {A, D, M , 1}

$dest$ = {A, D, M, MD, A, AM, AD, AMD, null}

Exercise: In small groups, implement the following tasks using Hack:

- sum = 0
- j = j + 1
- q = sum + 12 – j
- arr[3] = -1
- arr[j] = 0
- arr[j] = 17
- etc.

Symbol table:

j	3012
sum	4500
q	3812
arr	20561

(All symbols and values are arbitrary examples)

THE UNIVERSITY of ADELAIDE

Coding examples

Implement the following tasks using Hack commands:

- goto 50
- if D==0 goto 112
- if D<9 goto 507
- if RAM[12] > 0 goto 50
- if sum>0 goto END
- if x[i]<=0 goto NEXT.

Hack commands:

A-command: @value // set A to value

C-command: dest = comp ; jump // dest = and ;jump // are optional

Where:

comp = 0 , 1 , -1 , D , A , !D , !A , -D , -A , D+1 , A+1 , D-1, A-1 , D+A , D-A , A-D , D&A , D|A , M , !M , -M ,M+1, M-1 , D+M , D-M , M-D , D&M , D|M

dest = M , D , MD , A , AM , AD , AMD, or null

jump = JGT , JEQ , JGE , JLT , JNE , JLE , JMP, or null

In the command dest = comp; jump, the jump materialzes if (comp jump 0) is true. For example, in D=D+1,JLT, we jump if D+1 < 0.

Hack convention:

- True is represented by -1
- False is represented by 0

Symbol table:

sum	2200
x	4000
i	6151
END	50
NEXT	120

(All symbols and values in are arbitrary examples)

IF logic – Hack style

High level:

```
if condition
{
   code block 1
}
else
{
    code block 2
}
code block 3
```

Hack:

```
    D ← not condition
    @IF_TRUE
    D;JEQ
    code block 2
    @END
    0;JMP
(IF_TRUE)
    code block 1
(END)
     code block 3
```

Hack convention:

- True is represented by -1
- False is represented by 0

WHILE logic – Hack style

High level:

```
while condition
{
    code block 1
}
Code block 2
```

Hack:

```
(LOOP)
      D ← not condition)
      @END
      D;JEQ
      code block 1
      @LOOP
      0;JMP
(END)
      code block 2
```

Hack convention:

- True is represented by -1
- False is represented by 0

Complete program example

C language code:

```
// Adds 1+...+100.
 into i = 1;
 into sum = 0;
 while (i <= 100)
{
    sum += i;
    i++;
 }
```

Hack assembly convention:

- Variables: lower-case
- Labels: upper-case
- Commands: upper-case

Hack assembly code:

```
// Adds 1+...+100.
      @i     // i refers to some RAM location
      M=1    // i=1
      @sum   // sum refers to some RAM location
      M=0    // sum=0
(LOOP)
      @i
      D=M     // D = i
      @100
      D=D-A   // D = i - 100
      @END
      D;JGT   // If (i-100) > 0 goto END
      @i
      D=M     // D = i
      @sum
      M=D+M   // sum += i
      @i
      M=M+1   // i++
      @LOOP
      0;JMP   // Got LOOP
 (END)
      @END
      0;JMP   // Infinite loop
```

This Week

- Review Chapters 4 & 5 of the Text Book (if you haven’t already)
- Assignment 2 Due
- Start Assignment 3
- Prac Exam; details in announcement
- Review Chapter 6 of the Text Book before next week.