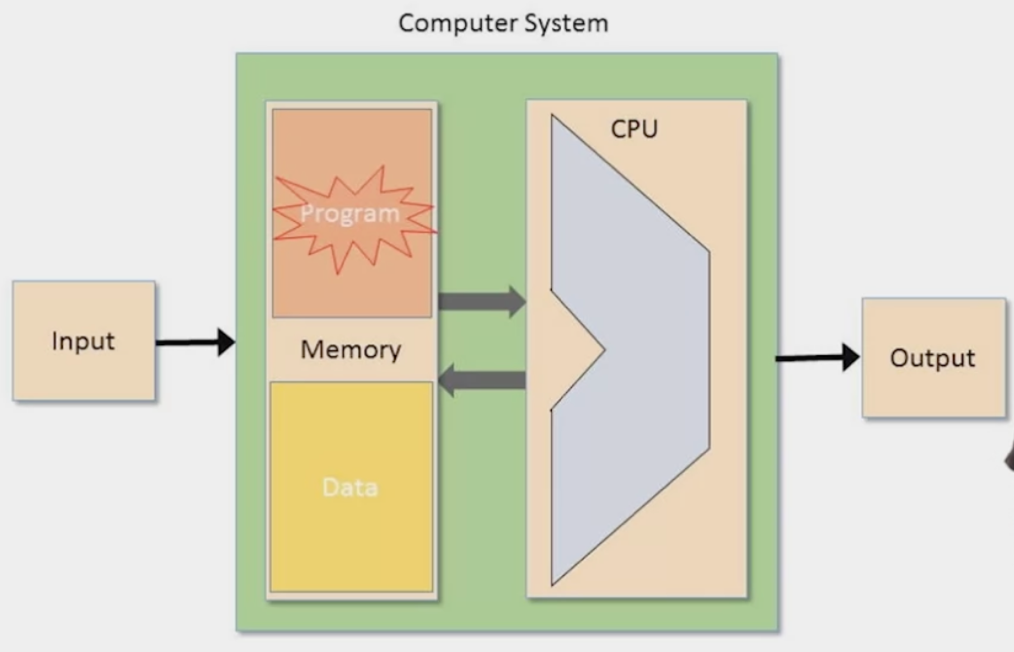
**Module 4: Machine Language Roadmap**

Unit 4.1: Machine Languages: Overview

Stored Program Computer



1. Machine Language: Operations, Program Counter, Addressing
2. **Mnemonics:**

Interpretation 1:

The “symbolic form” doesn’t really exist but is just a convenient mnemonic to present machine language instructions to humans

Interpretation 2:

We will allow humans to write machine language instructions using this “assembly language” and will have an “Assembler” program convert it to the bit-form

Unit 4.2: Machine Languages: Elements

1. Machine Operations

Usually correspond to what’s implemented in Hardware

* Arithmetic: Operations: add, subtract, ..
* Logical Operations: and, or…
* Flow Control: “goto instruction X”, “if C then goto instruction Y”

Difference between machine languages

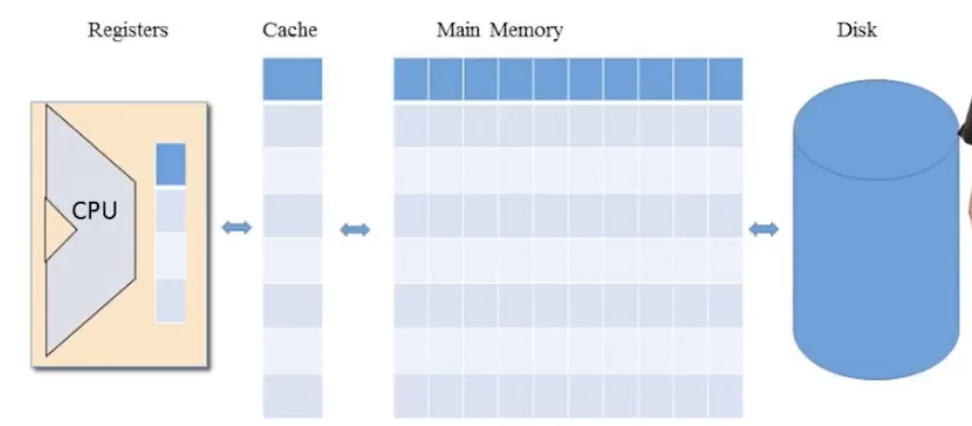
* Richness of the set of operations
* Data types (width, floating point)

1. Memory Hierarchy

Accessing a memory location is expensive

* Need to supply a long address
* Getting the memory contents into the CPU takes time

Solution: Memory Hierarchy



1. Registers

CPUs usually contains a few, easily accessed registers

Their number and functions are a central part of the machine language

Data Registers: e.g.: Add R1, R2

Address Registers: Store R1, @A

1. Addressing Modes

Register: Add R1, R2 // R2 <= R2 + R1

Direct: Add R1, M[200] // Mem[200] <= Mem[200] + R1

Indirect: Add R1, @A // Mem[A] <= Mem[A] + R1

Immediate: Add 73, R1 // R1 <= R1 + 73

1. Input/Output

* Many types of input and output devices

keyboard, mouse, camera…

* CPU needs some kind of protocol to talk to each of them

software “Drivers” know these protocols

* One general method of interaction uses “memory mapping”

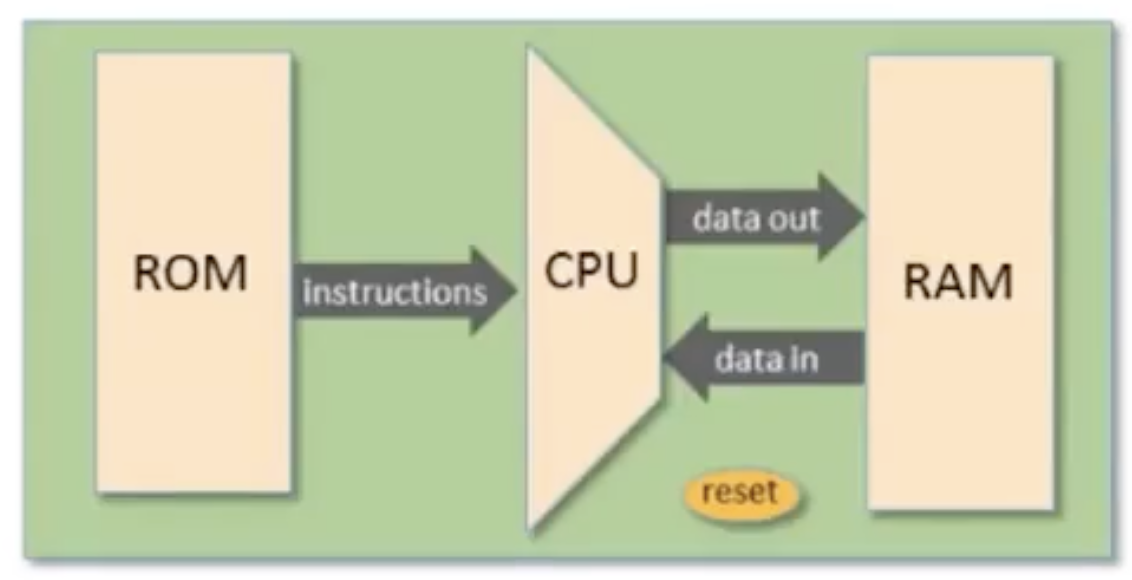
Memory Location 12345 holds the direction of the last movement of the mouse

Memory location 45678 is not real memory location but a way to tell the printer which paper to use

1. Flow Control

* Usually the CPU executes machine instructions in sequence
* Sometimes we need to “jump” unconditionally to another location,   
  e.g. so we can loop
* sometimes we need to jump only if some condition is met

Unit 4.3: The Hack Computer and Machine Language



A 16-bit machine consisting of:

* Data Memory(RAM): a sequence of a 16-bit registers: RAM[0], RAM[1]
* Instruction memory(ROM): a sequence of 16-bit registers: ROM[0], ROM[1]
* Central Processing Unit: performs 16-bit instruction
* Instruction bus/ Data bus/ Address buses

Hack programs:

sequence of instructions written in hack machine languages

Controls:

* The ROM is loaded with a Hack program
* The reset button is pushed
* The program starts running

The Hack machine language recognizes three registers:

* D holds a 16-bit value // D for Data
* A holds a 16-bit value // A for Addressing
* M represents the 16-bit RAM register addressed by A // M for Memory

**The A-instruction**

Syntax: @value where value is either:

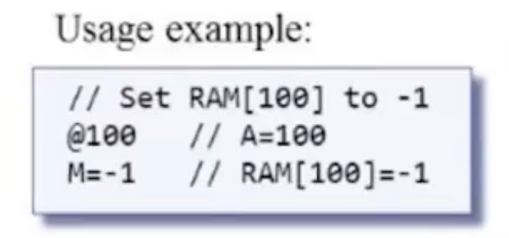
a non-negative decimal constant or a symbol referring to such a constant

Semantics:

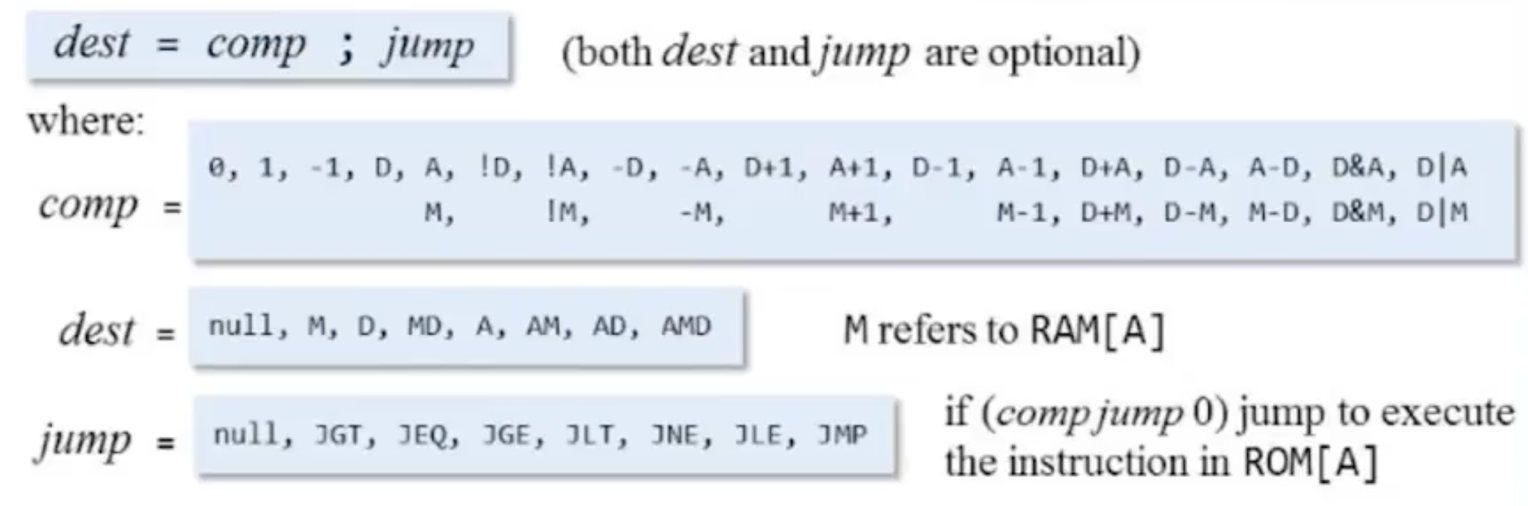
* Sets the A register to value
* side effect: RAM[A] becomes the selected RAM register

Example

@21: set A register to 21, and RAM[21] become the selected RAM register

 M refers to RAM[A]

**The C-instruction**



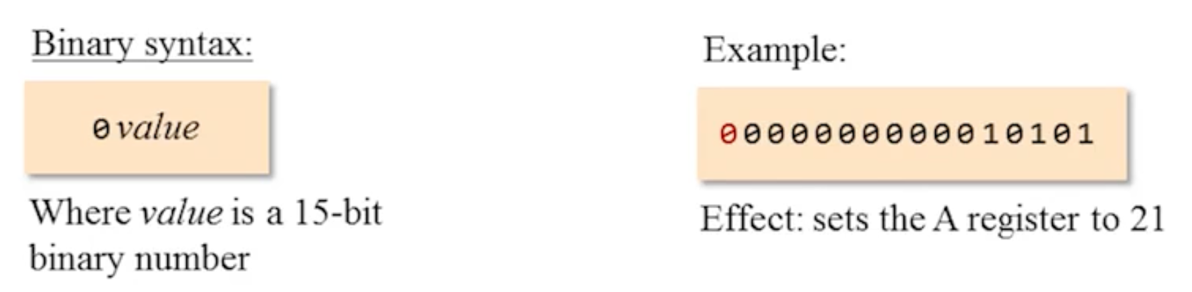
null means we don’t want to store the result of the computation

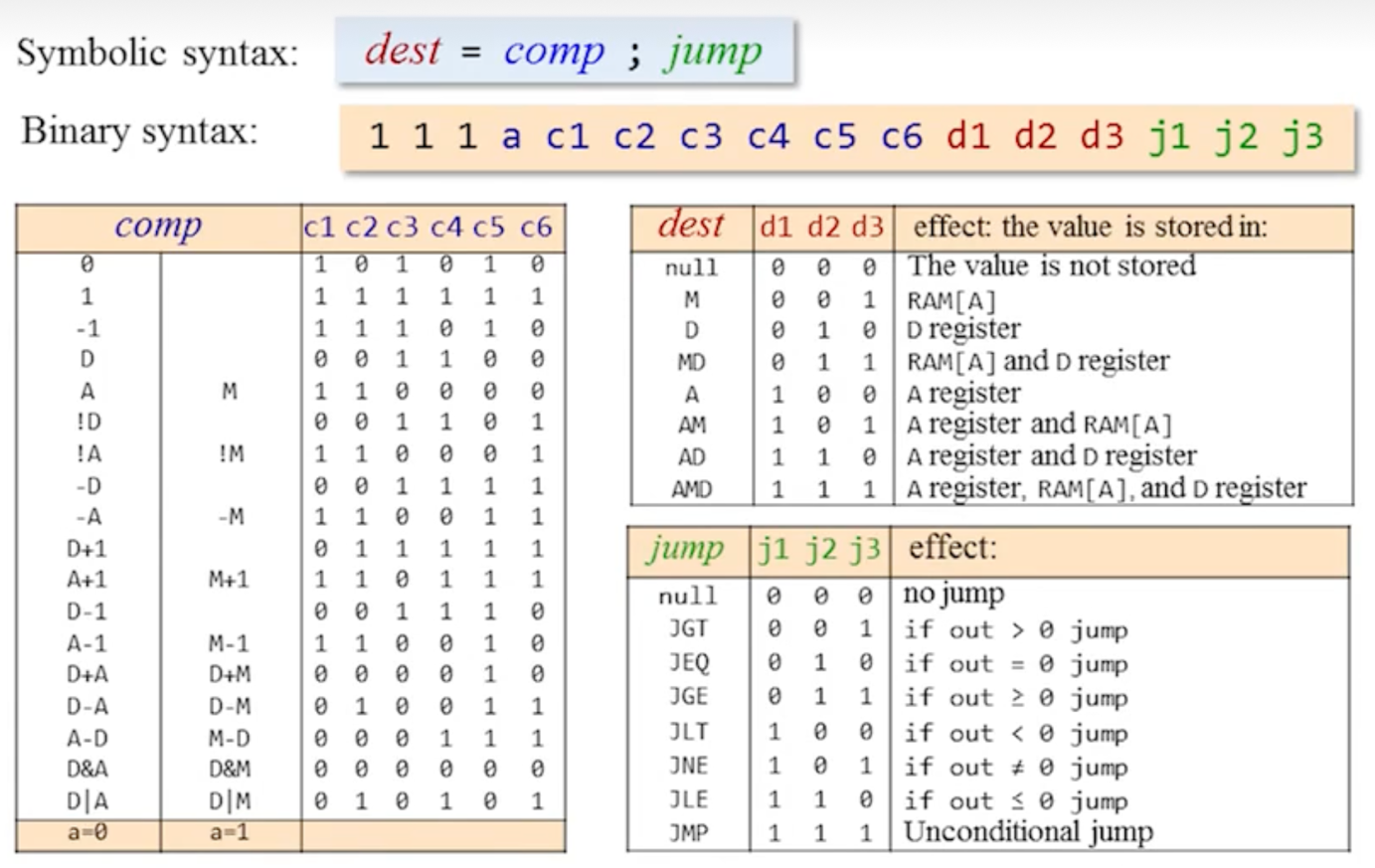
Semantics:

* Compute the value of *comp*
* Store the result in *dest*
* If the Boolenan expression (*comp jump* 0) is true

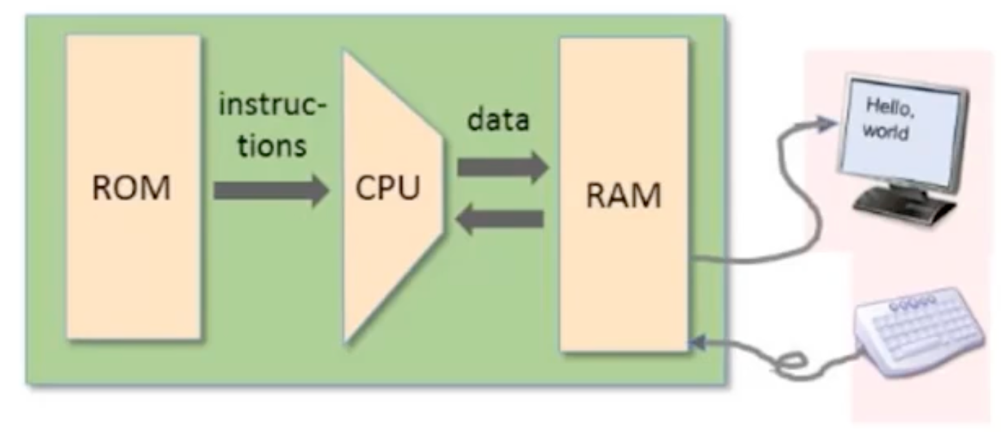
jumps to execute the instruction stored in ROM[A]

Unit 4.4: The Hack Language Specification





Unit 4.5: Input/Output



Peripheral I/O devices:

* Keyboard: used to enter inputs
* Screen: used to display outputs

Screen memory map

a designated memory area, dedicated to manage a display unit

the physical display is continuously refreshed from the memory map

The memory map is a sequence of 16-bit values .

Every one of these values is sometimes called a word

图示

描述已自动生成

To set pixel (row,col) on/off:

1. word = Screen[32\*row + col/16] // Screen is a chip

word = RAM[16384 + 32\*row + col/16]

1. Set the (col % 16)th bit of word to 0 or 1
2. Commit word to the RAM

To check which Key is currently pressed:

* Probe the contents of the keyboard chip
* In the Hack computer: probe the contents of RAM[24576]

**Unit 4.6 – 4.8: The Hack Programming Language:**

表格

描述已自动生成

The Hack assembly language features built-in symbols

* R0, R1, …, R15: “virtual registers”
* SCREEN and KBD: base addresses of I/O memory maps
* Remaining symbols: used in the implementation of the Hack virtual machine

discussed in Nand 2 Tetris pt.2

1. @LABEL translates to @n,

where n is the instruction number of following the (LABEL) declaration

Contract:

* Label declarations are not translated
* Each reference to a label is replaced with a reference to the instruction number following that label’s declaration

1. @temp: find some available memory register (say register n)

and use it to represent the variable temp. So, from now on. each occurrence of @temp in the program will be translated into @n

Contract

* A reference to a symbol that has no corresponding label declaration is treated as a reference to a variable
* Variables are allocated to the RAM **from address 16 onward**

1. Pointers:

* Variables that store memory addresses like arr and i are called pointers
* Hack pointer logic: whenever we have to access memory using a pointer, we need an instruction like A = M
* Typical pointer semantics:

set the address register to the contents of some memory register

1. Handling the keyboard

To check which key is currently pressed:

* Read the contents of RAM[24576]
* If the register contains 0, no key is pressed
* Otherwise, the register contains the scan code of the currently pressed key