i206: Lecture 4:
The CPU, Instruction Sets, and How Computers Work

Tapan Parikh

Spring 2013

Some slides courtesy Marti Hearst, John Chuang and others

Computer Memory

- We represent data as 0's and 1's (bits)
  - 8 bits in a byte
  - 2 to 4 bytes in a word
  - Words represented schematically as rows in memory
  - Each word has
    - High order bit (most significant bit)
    - Low order bit (least significant bit)
- We *also* represent locations as 0's and 1's
  - These locations are called addresses

Machine Language

- Only a very few unique instructions (commands) are needed for all CPU functionality.
- Called:
  - Machine instructions
  - Machine language
  - Machine code
- Represented in binary
- Assembly language: the human-readable version
  - Each statement of assembly language corresponds to one machine instruction.
- Need an assembler program to convert assembly language to machine code.

Machine Language

- Why so few instructions?
- (Nearly) everything is built up from:
  - Add/subtract/compare values (for jumps)
  - Load (into registers from memory)
  - Store (from registers into memory)
  - Get next instruction (or jump to another instruction)
- Also referred to as arithmetic, memory and control instructions

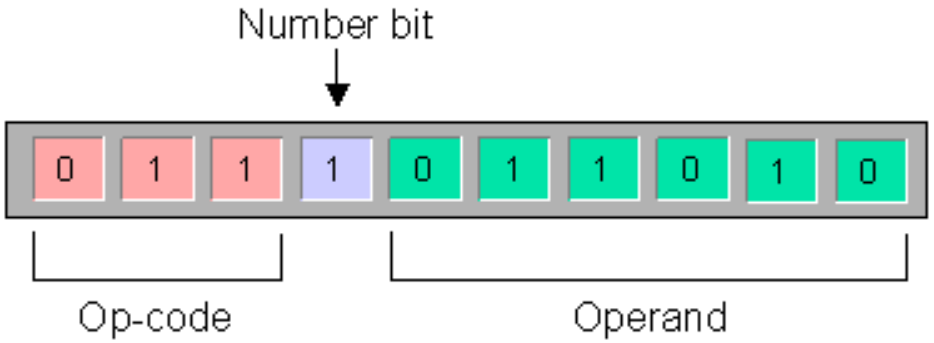
Relation to High Level Languages

- Java, Python, C are all high level programming languages.
- They are translated to assembly language by a compiler and/or an interpreter.

The Stored-Program Concept

- Programs (code) are represented in the same way that data is.
- This way, both code and data can be stored in the same place as well.
- This is a key concept for understanding how the CPU works.

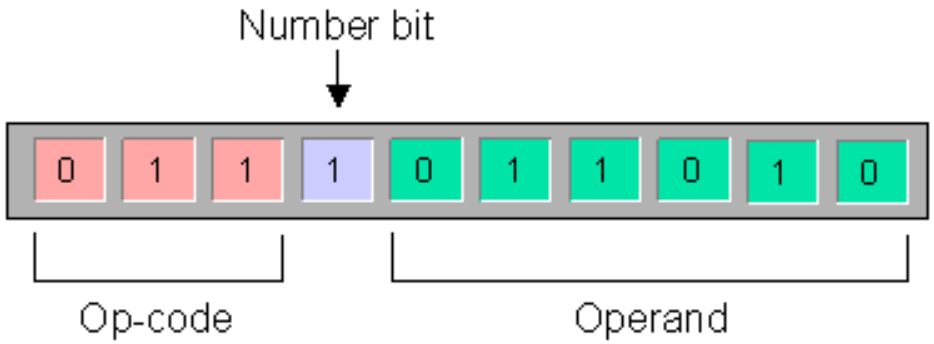
Instructions and Their Representation



The op-code area holds operations.
How many unique op-codes can this setup support?

Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

Instructions and Their Representation

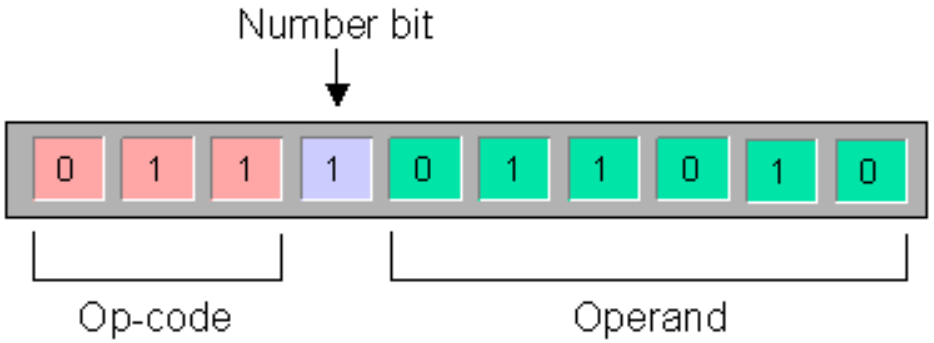


Op-code	*Mnemonic*	*Function*	*Example*
001	LOAD	Load the value of the operand into the Accumulator	LOAD 10
010	STORE	Store the value of the Accumulator at the address specified by the operand	STORE 8
011	ADD	Add the value of the operand to the Accumulator	ADD #5
100	SUB	Subtract the value of the operand from the Accumulator	SUB #1
101	EQUAL	If the value of the operand equals the value of the Accumulator, skip the next instruction	EQUAL #20
110	JUMP	Jump to a specified instruction by setting the Program Counter to the value of the operand	JUMP 6
111	HALT	Stop execution	HALT

A simple machine language

Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

Instructions and Their Representation



The Operand area holds memory addresses and data values.

The number bit tells us if its address or value

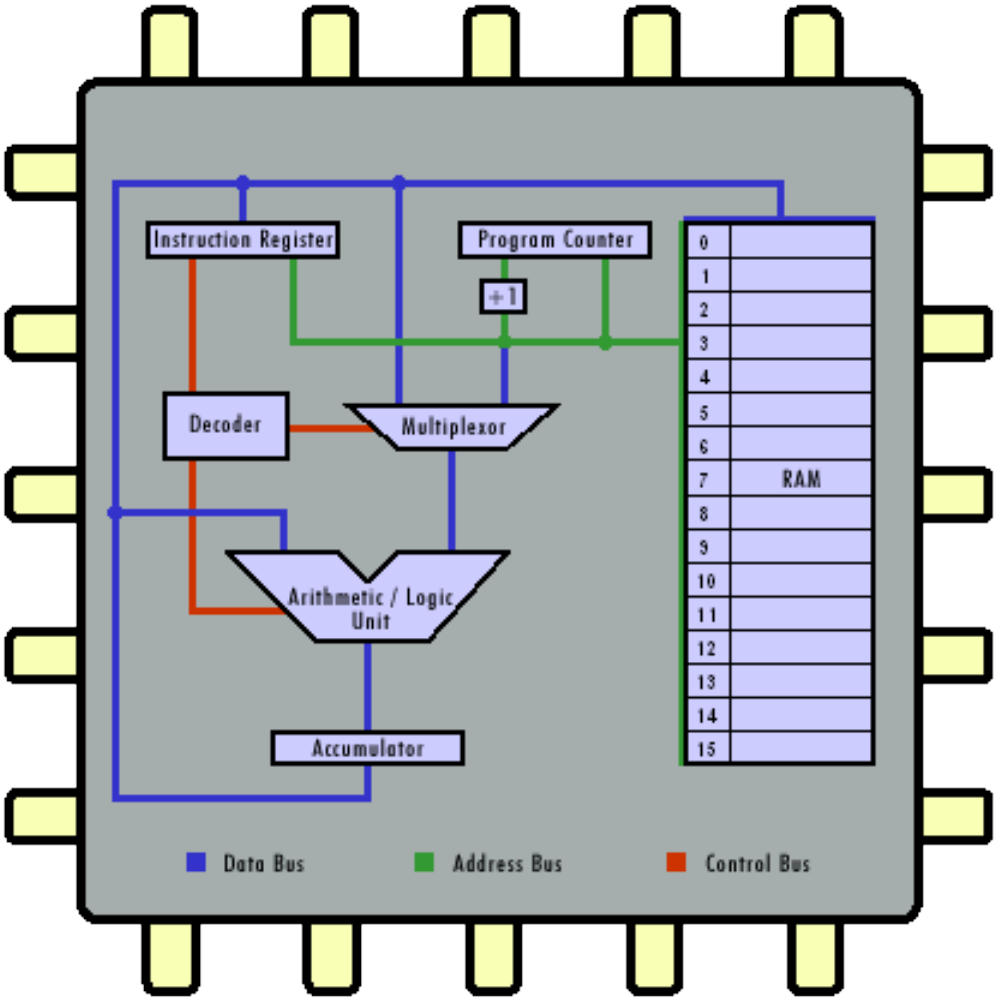
What is the largest memory address?

111111 = 1 + 2 + 4 + 8 + 16 + 32 = 63

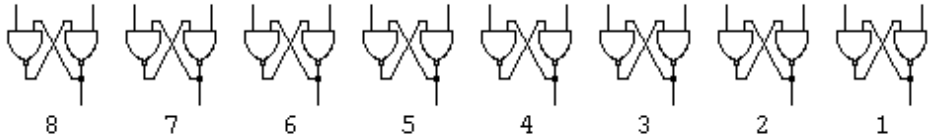
Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

The CPU

Central Processing Unit



The CPU



- Registers
  - Are basically rows of latches
  - Store values
  - Can be used to hold values of variables as well as memory addresses and code.
- Special registers
  - PC (program counter)
    - Points to the location in memory that has the current instruction.
  - Instruction Register
    - Holds the current instruction.
  - Accumulator
    - Hold the current value of the ALU

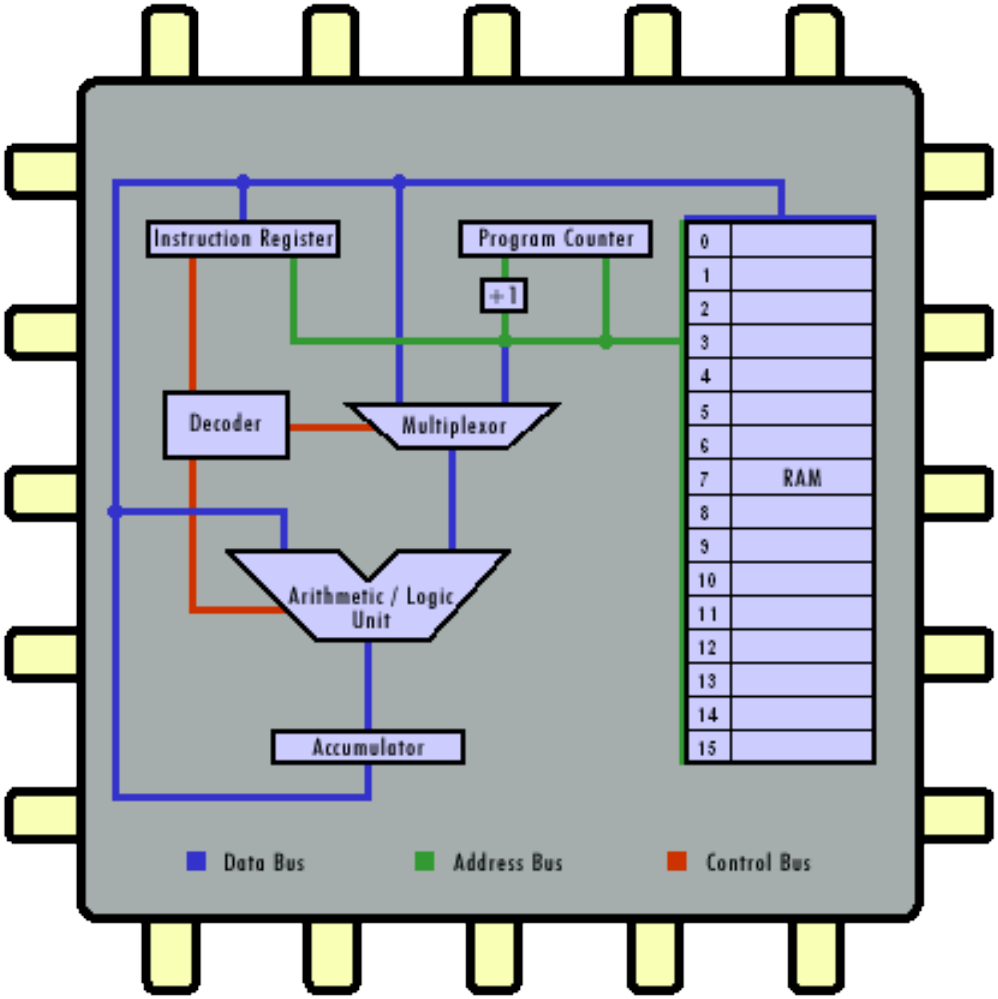
The CPU

Bus: bundles of wires connecting things.

Instruction Register: holds instructions.

Program Counter: points to the current memory location of the executing program.

RAM: stores the code (instructions) and the data.



Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

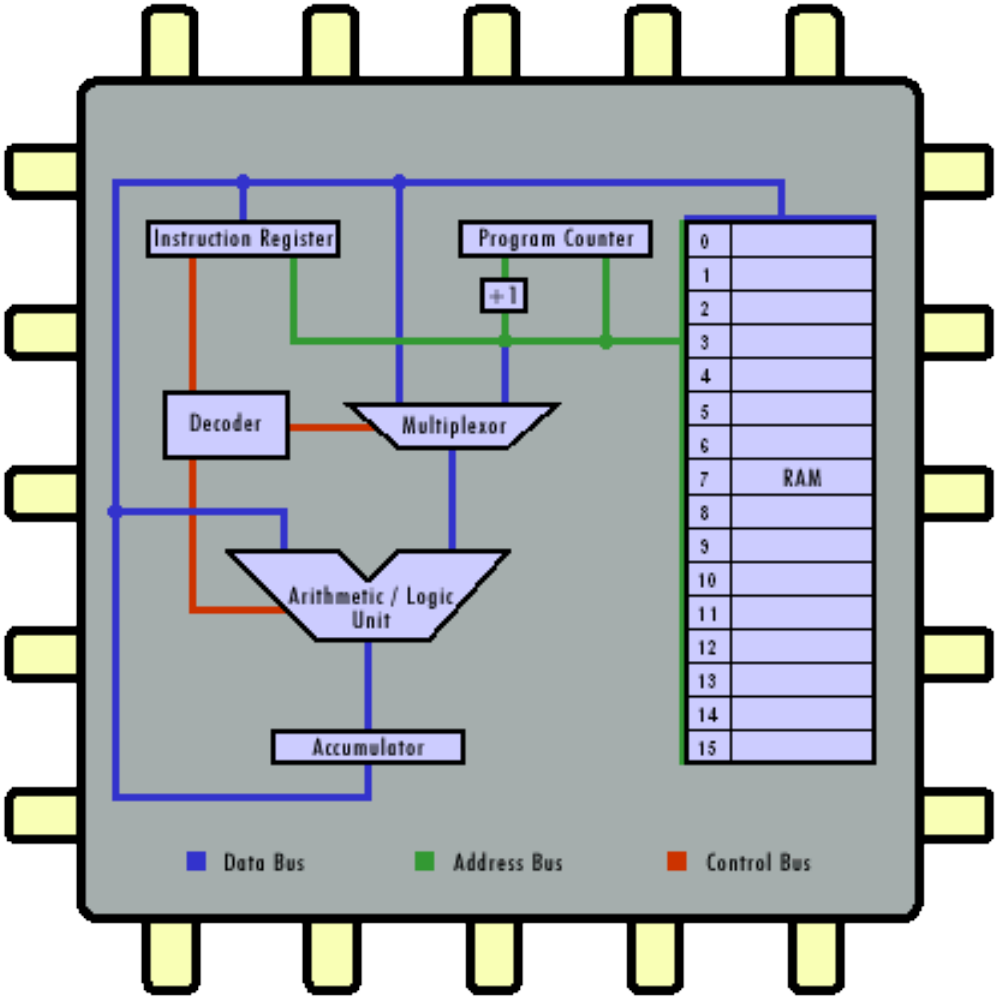
The CPU

MUX: multiplexer –
chose 1 out of N (ALU or RAM in this case)

ALU: adds, subtracts, compares two numbers (it has 2 words of storage, on the left and the right)

Accumulator:
stores ALU's output

Decoder: uses the 3-bit op-code and the number bit to control the MUX and the ALU and the memory buses.



Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

Machine code to add two numbers that are stored in memory (locations 13 and 14) and then store the result into memory (location 15).

0	001 0 001101	LOAD	13	Load the value of memory location 13 into the Accumulator
1	011 0 001110	ADD	14	Add the value of memory location 14 to the Accumulator
2	010 0 001111	STORE	15	Store the value of the Accumulator in memory location 15
3	111 0 000000	HALT		Stop execution

The Program Counter starts at memory location 0.

Image adapted from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

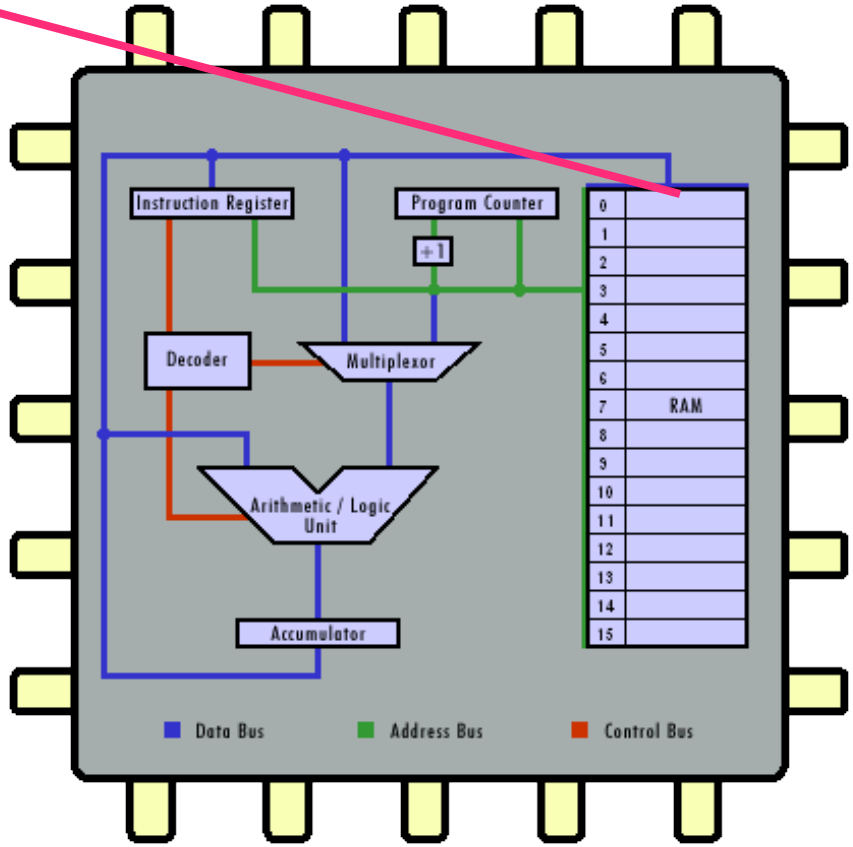
0	001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator

The Program Counter starts at memory location 0.

(Notice: the instructions and the data are both stored in the RAM.)

This instruction moves along the bus into the instruction register.

The decoder checks the number bit to see how to handle the op-code.



Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

0	001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator

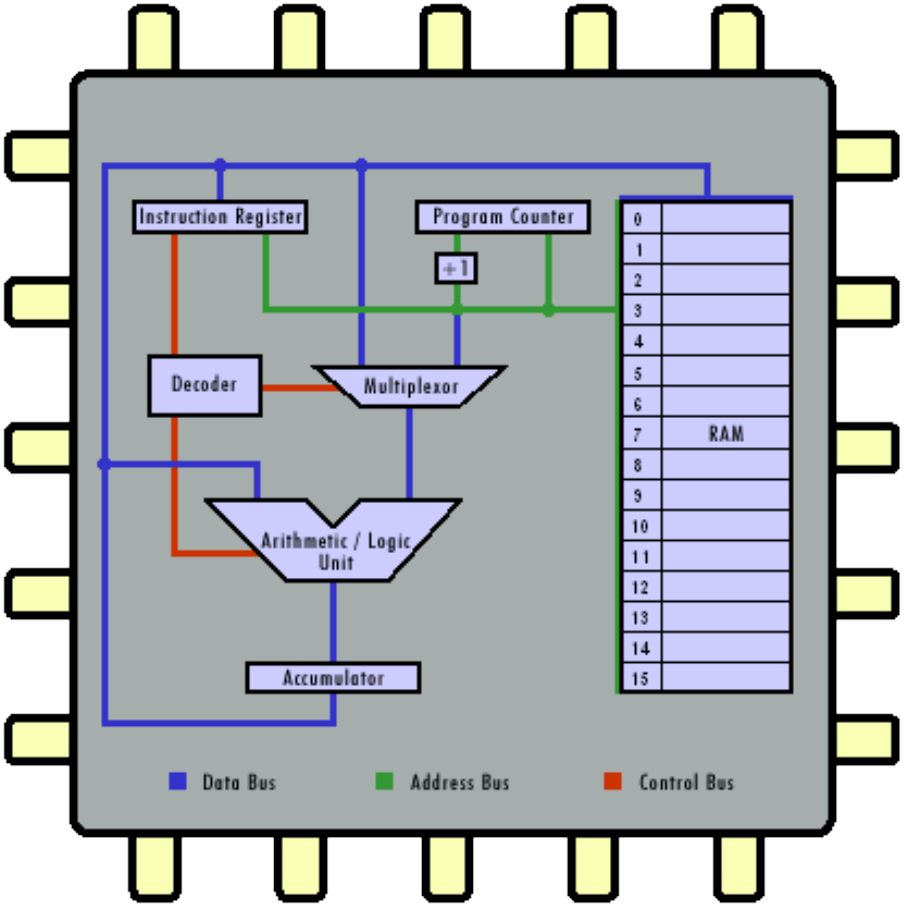
If the op-code number bit = 0, we want the operand to be used as a memory address.

So the decoder switches the MUX to connect the operand to the RAM.

If the bit were 1, the MUX would connect the operand to the ALU.

The green wires do the addressing of the RAM.

The blue wires do the reading to and writing from the RAM.



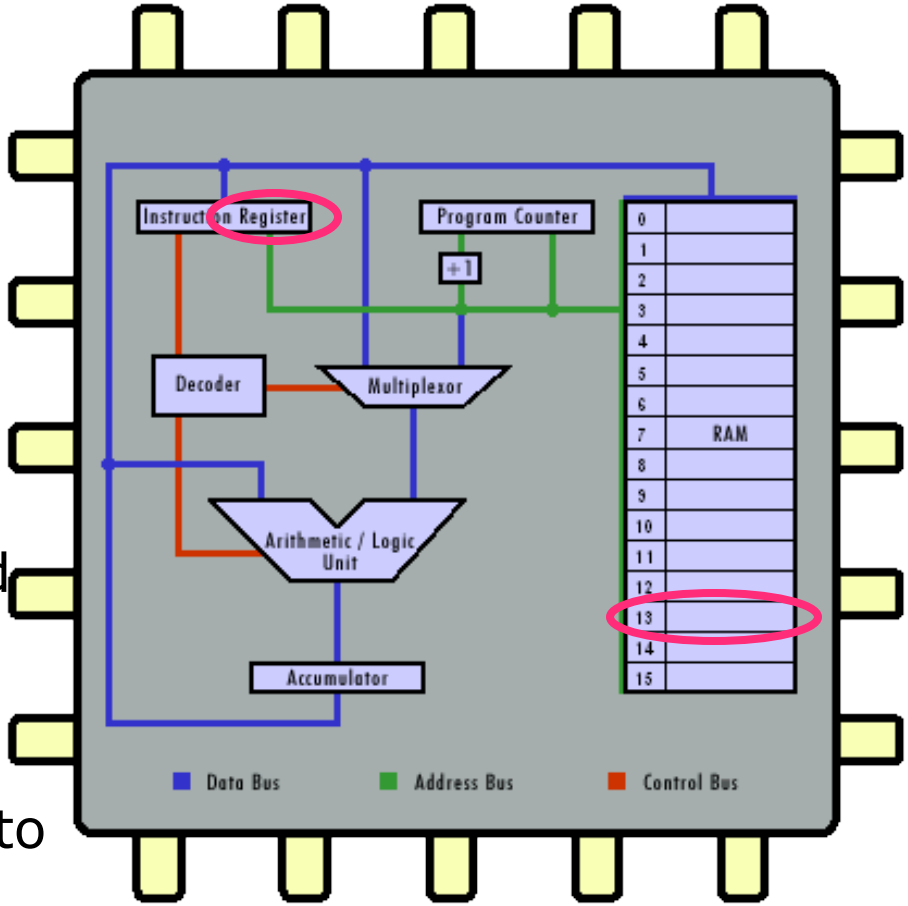
Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

0	001 0	001101	LOAD 13	Load the value of memory location 13 into the Accumulator

The Decoder tells the ALU this is a LOAD instruction.

The memory location 13 is activated, and its value placed on the blue bus. This goes through the multiplexer into the ALU. It also trickles down into the Accumulator (Load is implemented as an ADD value to 0).

The last step (always, except for a jump) is for the Program Counter to get incremented. Now it points to memory location 1.



Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

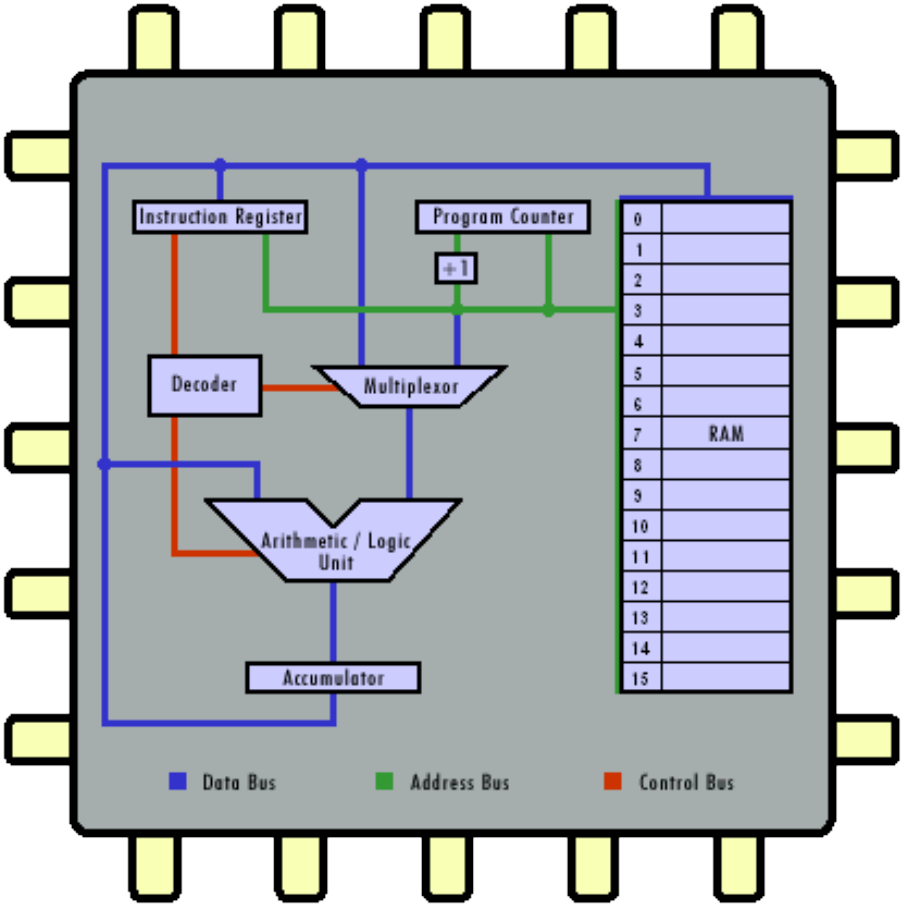
1	011 0 001110	ADD	14	Add the value of memory location 14 to the Accumulator

Instruction: ADD 14
Similar to Load 13, except the ALU is given the ADD operation.

The contents of the Accumulator are fed into the left hand operand, and the contents of memory location 14 becomes the right hand operand.

Results of the AND end up in the Accumulator.

The Program Counter increments.



Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

2	010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15

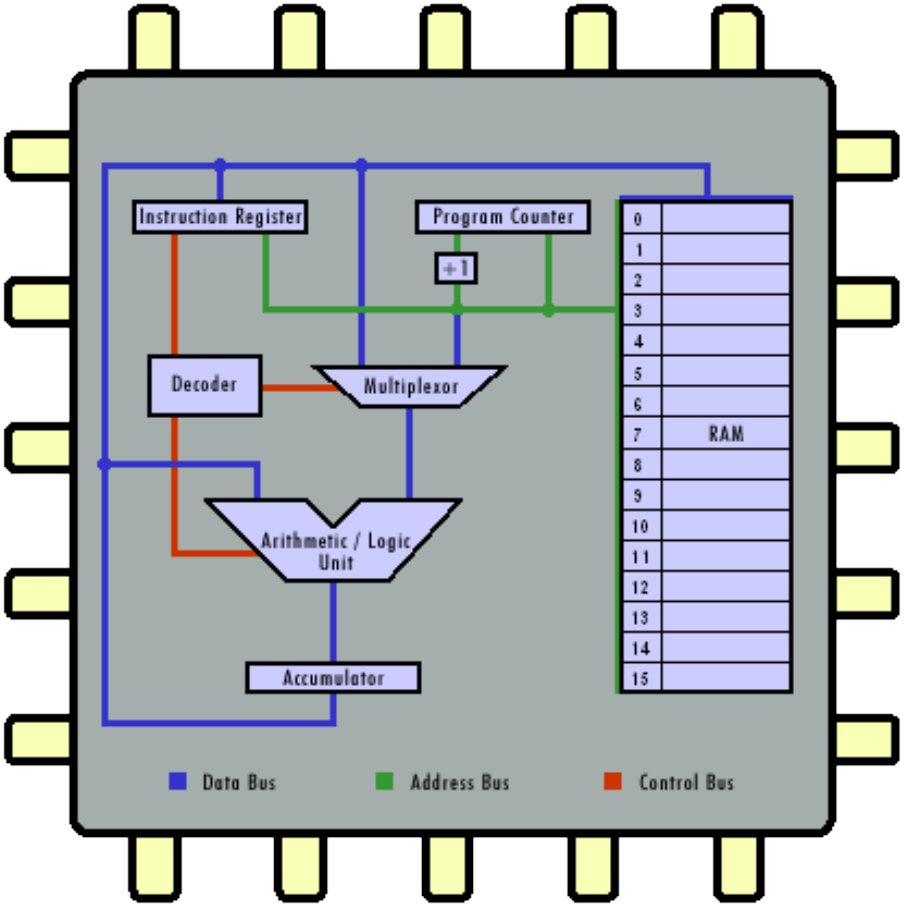
Instruction: STORE 15

Memory address 15 is activated via the green bus.

The contents of the Accumulator are written to RAM in location 15 using the blue bus.

The Program Counter increments.

The next instruction is HALT.



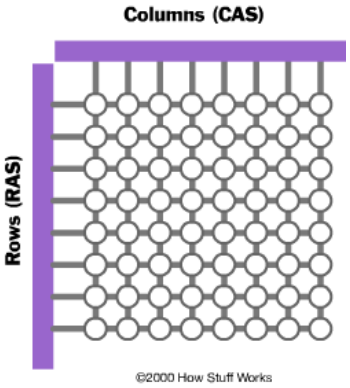
Images from http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/

CPU Performance

- The number of INSTRUCTIONS per operation plays a big role.
  - Each operation requires some instructions

    e.g., about 6 instructions required for the loop part of a simple for-loop.
  - Each instruction takes some number of cycles to execute.
  - So … a CPU with an instruction set and circuit design that requires fewer cycles to get operations done might have better performance than a CPU with a faster clock but more complicated circuitry.

RAM
(Random Access Memory)



- One set of wires activates the appropriate location
- Another set of wires
  - reads the data value from that location or
  - writes the data value to that location
- Read: get the value (don't change it)
- Write: set the value (change it)
- RAM is arranged as a grid
  - Each column is a word (or some number of bytes)
  - First activate the correct column
  - Then either
    - Read the word's values into the row output, or
    - Write the row's values into the word

How DRAM Works

Saw we want to set this memory address to contain the value 01001011
When two activated lines cross, the cell at the intersection gets turned on.

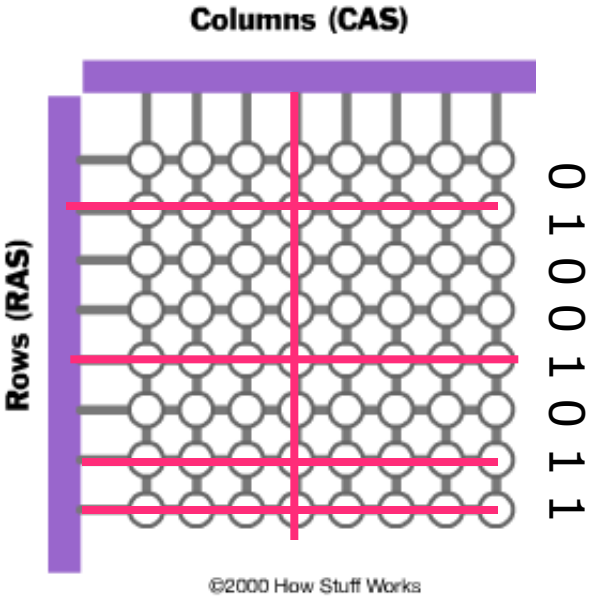


Image adapted from http://www.howstuffworks.com/ram.htm

How DRAM Works

Now we've set the value.
Reading moves the data the other direction, onto the row (a special control wire, not shown, tells the circuit whether to read or write).

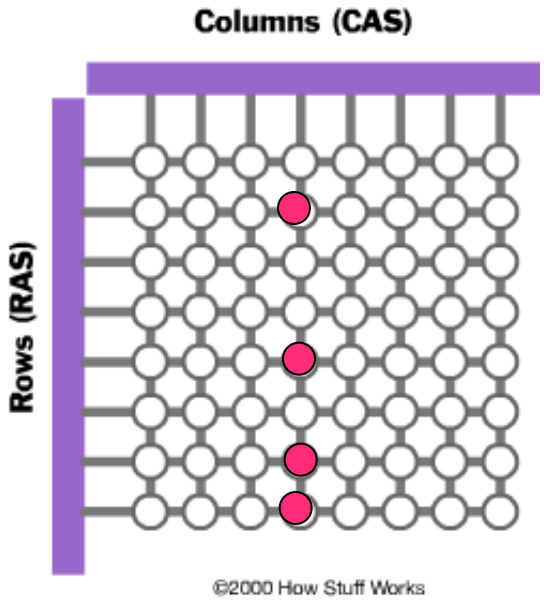


Image adapted from http://www.howstuffworks.com/ram.htm

Measuring CPU Performance

- MIPS:
  - **m**illions of **i**nstructions **p**er **s**econd
  - "meaningless indicator of performance" (a joke)
  - Some instructions require more time than others
  - No standard way to measure MIPS
- FLOPS:
  - **fl**oating-point **op**erations per **s**econd
  - Measures the speed of the floating point unit
    - A special circuit that does math on real numbers
    - So, doesn't measure performance on non-numeric applications
- Benchmark:
  - A test used to compare performance of hardware and/or software.
  - Different benchmarks measure different aspects of performance
- SPEC:
  - **S**tandard **P**erformance **E**valuation **C**orporation
  - A set of standardized tasks (benchmarks) that CPUs are run on in order to assess their performance.

Adapted form http://webopedia.internet.com/

CPU Performance

CPU clocks are related to another commonly used term…

Real Time

- Means regular clock time as opposed to CPU clock time
- Other factors also prevent a program from running in real time
  - CPU scheduling, multi-tasking

In common usage today,

- Real time means right now, as I'm talking, as opposed to
- Offline, when I have to think on my own, or when we're not in the middle of a timed event with others listening.

Summary

- Boolean logic is used to make logic gates, which build up into circuits.
- Binary is used to represent both code and data in memory, in instruction registers, and in the arithmetic logic unit.
- Binary instructions move through the circuits, controlling the computer's operation and actions and data flow.
  - Adding, subtracting, storing, loading.
- This is pretty much what it takes to do computation!
  - After you write the program, compile it, and load it into the CPU and memory.