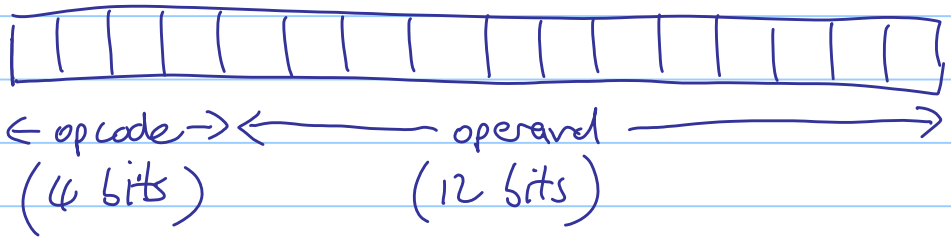
Assembly language introduction

Note Title

Instruction set architecture (ISA)
- specifies the machine language of a CPU
- includes
  - the format of the instructions
  - a description of what each instruction does

Typically, any machine instruction contains
- some bits that represent an opcode (a number identifying what operation the instruction performs
- some bits representing the arguments (or operands) for the instruction.

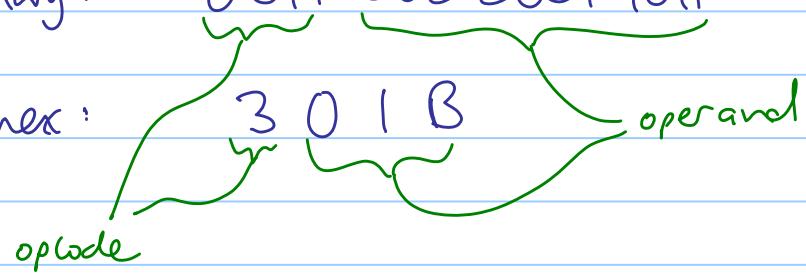
e.g. on the MARIE architecture introduced below, the format is:



the opcode for ADD is 3 (or 0011 in 4-bit binary) and ADD take a single operand that is a 12-bit address. So the instruction ADD 27 is:

in binary: 0011 000 0001 1011

in hex: 3 0 1 B



The meaning of "ADD 27" is defined to be "add the value in memory location 27 to the current value in the accumulator register, and store the result in the accumulator register."

Common instructions include LOAD, STORE, JUMP, ADD, MUL.

JUMP: jump to another location in the program

MUL: multiply

We study the ISA of a very simple computer called MARIE.

MARIE:
- has 16-bit words
- memory is word-addressable, and contains 16K words (i.e. 4096 words).
- instructions consist of a 4-bit opcode and a single 12-bit operand

[Demo: use simulator to see LOAD, STORE, ADD, JUMP
- SimpleAdd.mas and SimpleAddJump.mas show these
- show all relevant registers on whiteboard, trace contents]

Activity: type in and execute the following program:

```
load  0
store 5
halt
```

- can you explain the result? What does this reveal us about the key insight behind the von Neumann architecture?

Answer: instructions and data are indistinguishable - both are stored in memory. Any memory location can be regarded as instruction, data, or both.

Exercise:

a) Convert LOAD 03E into (i) hex
(ii) binary

b) Convert the machine language instruction hex 4105 into assembly.

Solutions a) i) 103E
ii) 0001 0000 0011 1110
b) SUBT 105

Instructions often require several micro-operations.
The micro-operations needed for a particular instruction are written in register transfer language (RTL).
See textbook section 4.8.4 for examples.
The details of the fetch-decode-execute cycle are given in book figure 4.11

Minilab: modify SimpleAdd.mas to add 5 numbers of your choosing and store the result.

Challenge exercise: Write out the RTL for a hypothetical instruction that doubles the contents of a given memory location