Topic for today:

Instructions and
Instruction Set Architecture (ISA)

General purpose registers

Many registers have dedicated purposes. Registers which may be accessed by the programmer to store arbitrary data are general purpose registers. If there is only one general purpose register, it is called the accumulator. (Compare this to the memory of a typical low-end hand-held calculator.)

Instructions

An instruction is one line in a program as it is stored in memory. Typically it is one word of memory. It is loaded into the Instruction Register and then executed. It can require one or (usually) more clock cycles to execute.

Instruction Set Architecture

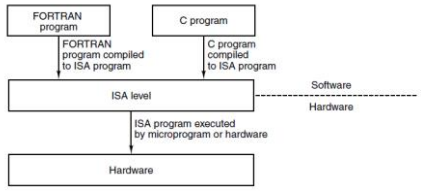
The *instruction set architecture* (*ISA*) of a computer specifies the instructions that the computer can perform. Two machines can have dramatically different physical designs but the same instruction set architecture.

Computer architecture = hardware + ISA

Instruction Set Architecture

- Interface between software and hardware
- What does the programmer/compiler/ executable code "see"?
  - Main memory
  - Some registers
  - Basic instructions (arithmetic, data moves, …)

Instruction Set Architecture



Designing an ISA

- Ideal world
  - architects will talk to both the compiler writers and the hardware engineers to find out what features they want in the ISA level
  - If the compiler writers want something the engineers can't produce at a reasonable cost, it doesn't get included
  - If the engineers want to put in some nifty new feature that the compiler writers can't figure out how to generate code for it, it doesn't go in.

Designing an ISA

- Real world
  - Customers want:
    - Compatibility with its predecessor
    - To be able to run their old OS on it
    - Run all existing applications without modifications
  - Requires ISA engineers to
    - Keep the same ISA between models
    - Back the new ISA **backwards compatible**

What makes a good ISA?

1. Programmability
   - Easy to express programs efficiently?
2. Implementability
   - Easy to design high-performance implementations?
3. Compatibility
   - Easy to maintain programmability (implementability) as languages and programs evolve?

What instructions to include?

- Minimum set is sufficient, but inconvenient for programmers
- Extremely large set is convenient for programmers, but
- expensive to implement.
- Architect must consider additional factors
  - Physical size of processor
  - How the processor will be used
  - Power consumption
- The set of operations a processor provides represents a tradeoff among the cost of the hardware, the convenience for a programmer, and engineering considerations such as power consumption.

Representation of instruction set?

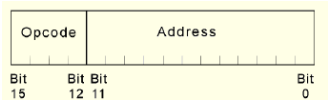
- Architect must choose
  - Set of instructions
  - Exact representation uses for each instruction (instruction format)
  - Precise meaning when instruction executed
- These items define the instruction set, specifying the format of its instructions and the primitive operations that the machine can perform.

What kinds of instructions?

- Processing data
- Moving data
- Controlling the execution sequence of a program.

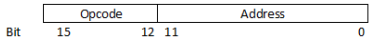
Instruction format

- An Instruction is represented as binary string
- Typically
  - Opcode at beginning of instruction
  - Operands follow opcode
- This is the format of a MARIE instruction

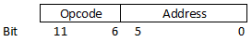


Instruction Length

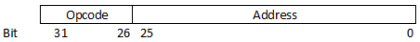
- Fixed Length
  - Ex. MARIE is a fixed length instruction set consisting of 16-bits



- Variable Length
  - 12-bits



  - 36-bits



Instruction length

- Fixed Length
  - Pro: Decodes faster
    - Less Complexity
  - Con: Wastes Space
    - Opcodes that do not require operands such as MARIE's *halt* makes no use of its address space.
    - Additionally, instructions must be **word aligned**. Creates **gaps** in memory

Instruction Length

- Variable Length
  - Pro: Saves storage space (Not exactly)
    - Instructions take up only as much space as needed
    - Instructions must be **word aligned** in main memory
    - Therefore instructions of varying lengths will create **gaps** in main memory
  - Con: Complex to decode

Address field

Depending on the type of instruction, the address field may contain:

- Nothing
- A constant
- The address of data to use in the operation
- The address of the address of data (i.e., a *pointer*)
- An offset to be added to a base address to compute the address of data

A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) and an address part (allowing for only one address). Each instruction is stored in one word of memory.

a) How many bits are needed for the opcode?

b) How many bits are left for the address part of the instruction?

c) What is the maximum allowable size for memory?

d) What is the largest unsigned binary number that can be accommodated in one word of memory?