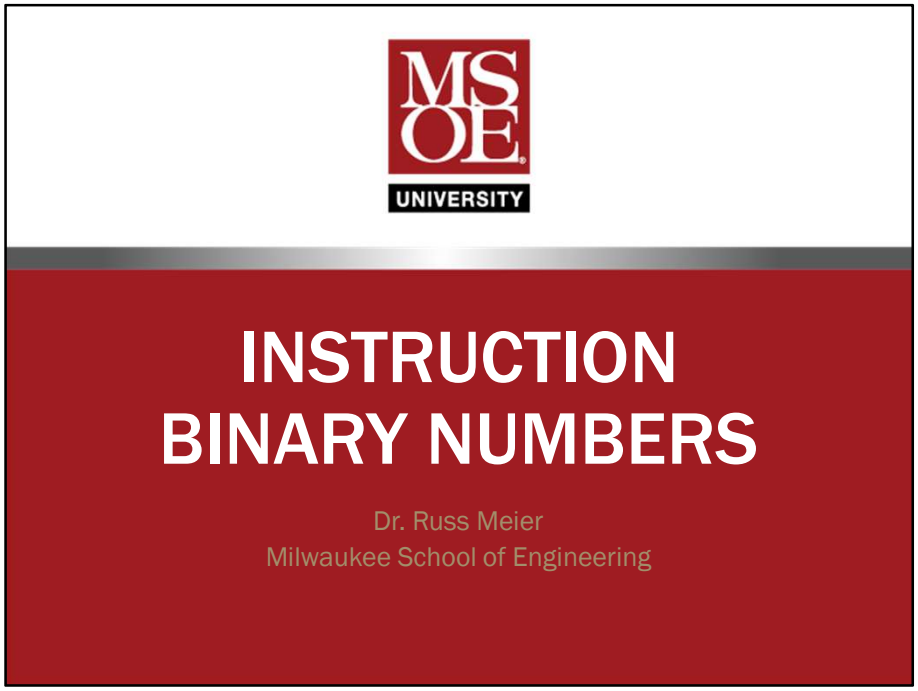
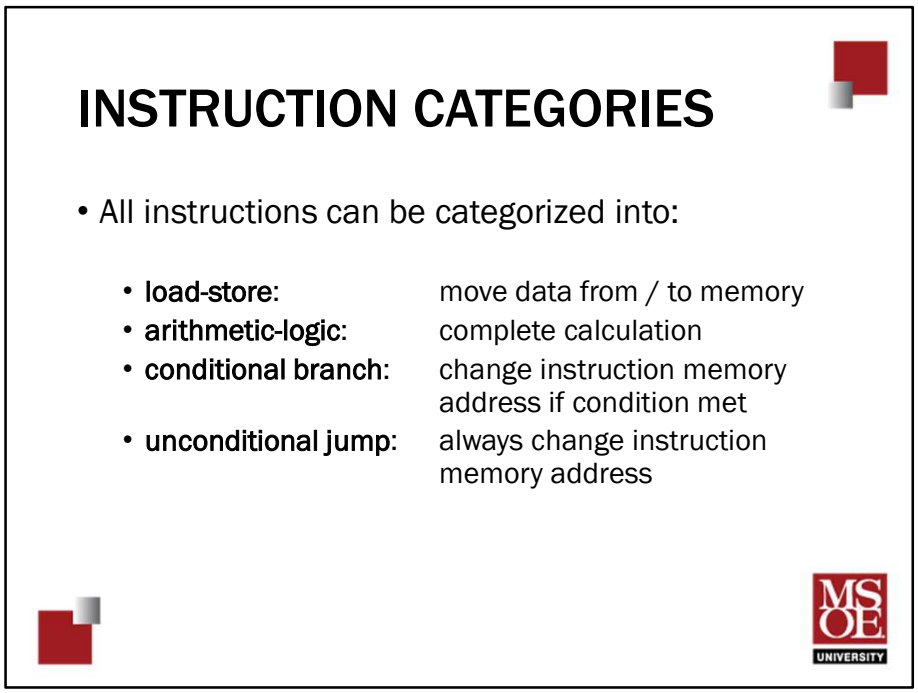


INSTRUCTION BINARY NUMBERS

Dr. Russ Meier
Milwaukee School of Engineering



All computer instructions can be categorized into four categories based on what type of operation they do.

- Memory load and store instructions move data between locations in the memory pyramid.
- Arithmetic and logic instructions calculate results from current pieces of data.
- Branch instructions change the instruction memory address to jump among instructions in a program.
- Some branches occur only when a certain arithmetic condition has occurred. For example, branch if the previous calculation produced zero. This type of branch instruction is called a **conditional branch**.
- Other branches always calculate a new instruction memory address. For example, programs jump to subroutines quite often. This doesn't depend on an arithmetic result. It just happens. This type of branch is called an **unconditional branch**.

ENCODING INSTRUCTIONS

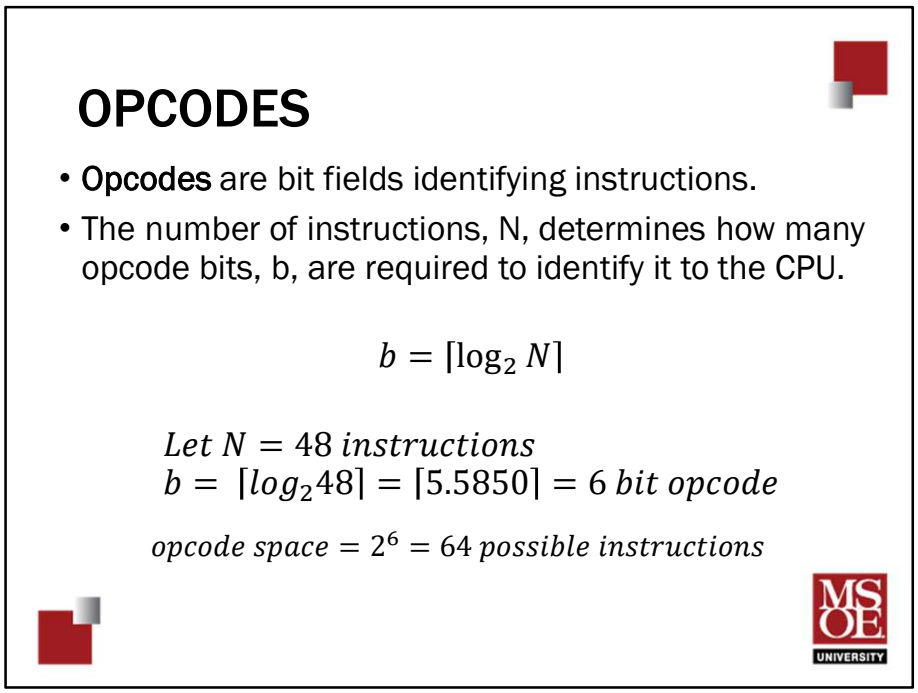
- **Instructions** are encoded as binary numbers.
- **Binary bit fields** encode parts of each instruction.
- **Opcodes** are bit fields identifying the instruction.
- **Operands** are bit fields identifying data locations.
- **Control bits** are bit fields providing control information.

OPCODE	SRC 1	SRC 2	DEST	SHAMT	FUNCT
000000	00101	01101	00111	00000	001101

0x00AD380D

Every machine instruction becomes a binary number stored in instruction memory.

- The binary number is separated into chunks or **fields** that contain information about the instruction.
- The field uniquely identifying the instruction as an add, a subtract, a multiply, a move, a shift, etc. is called the **opcode**.
- The opcode can be located anywhere within the instruction binary number but is often found at the most-significant end of the number or the least-significant end.
- The fields identifying the locations of data are called **operands**. The number of operands depends on the type of instruction set.
- Finally, control bits give information about the instruction to the CPU. For example, a control bit might command that the CPU should capture the ALU result flag.
- The fields form the entire binary number stored in instruction memory as voltages.

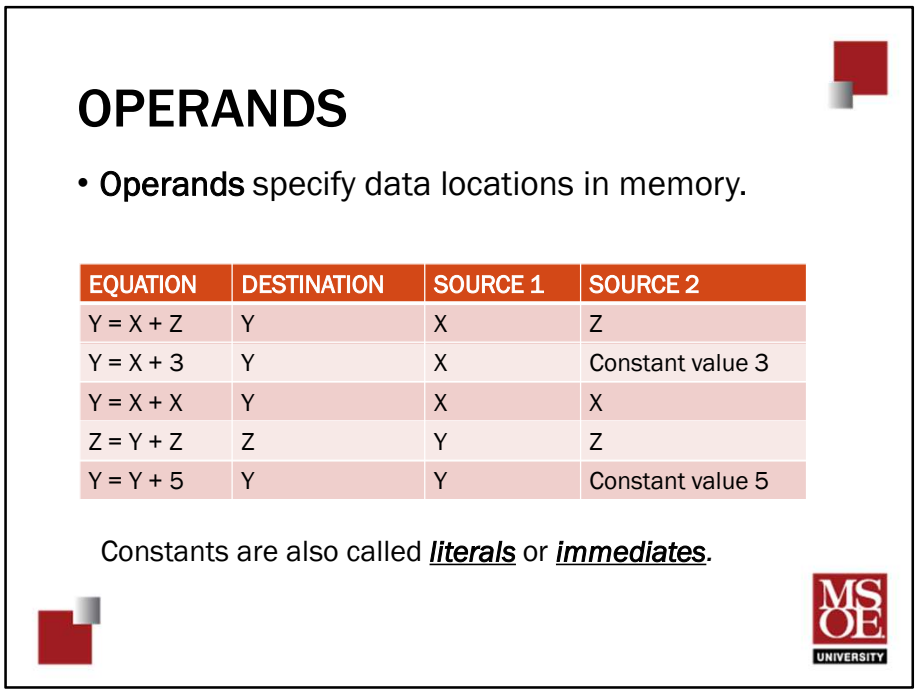


The number of instructions provided by a machine determines the size of the opcode bit field. Like all binary counting problems, the size is calculated using a base-2 logarithm. And, we round up the calculated number of bits because we don't electrically store fractions of a bit voltage in digital computers.

- This example shows the calculation for the opcode bit field size for a machine that provides 48 instructions. Five bits would allow 32 instructions total and 6 bits allows up to 64 instructions. Because 48 falls between 32 and 64, a 6-bit field must be used to ensure a complete set of unique binary opcodes.

- The opcode bit field defines the number line of opcode values. We call this number line the **opcode space**. In this example, the total number of possible instructions, and thus the size of the opcode space, is 64.

- Machines do not always use their complete opcode space. Often, opcodes are left **unused** or **reserved** to allow for future expansion of the instruction set.

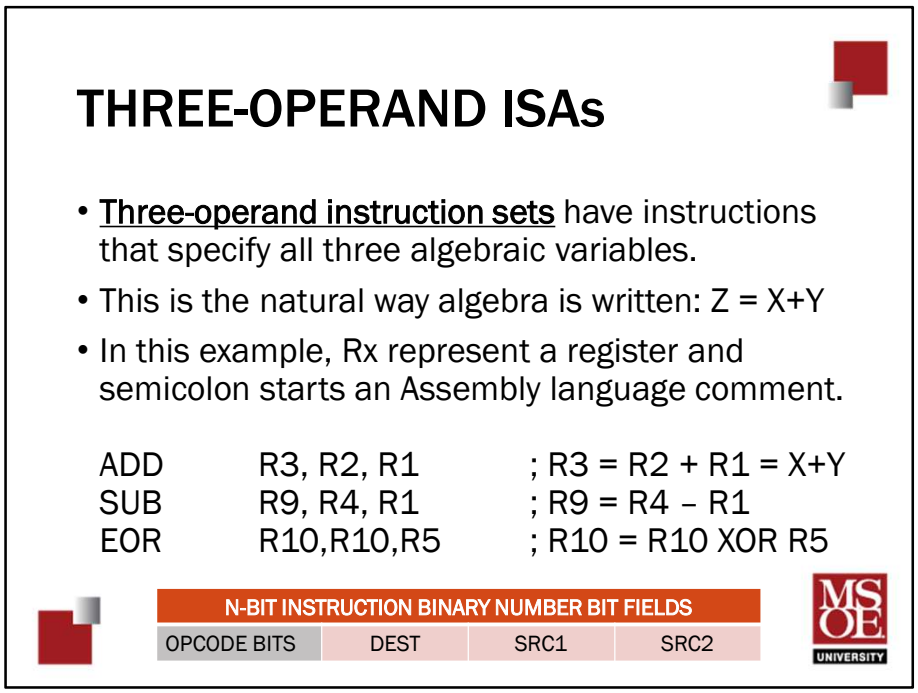


Basic arithmetic operations work on two values. Add, subtract, multiply, and divide all require two numbers at a minimum. These two numbers are called **operands** in computer architecture.

- The operand locations in memory must be encoded as part of the instruction binary number.
- In modern RISC instruction set architectures, the register file is the only location allowed to provide numbers to the ALU. Thus, for arithmetic instructions, the operand encodings will be numbers that specify what registers provide the numbers to the ALU. Load-store and branch instructions will have operands that specify memory addresses.

Let's look at the example equations in this table.

- Algebraically, we can think of the two memory locations using variable names. In the examples on this slide, variables X, Y, and Z are used.
- Sometimes, algebra includes constants. For example, the equation Y = X+3 in row two of the table uses the constant value 3.
- Constants are also known as **literals**, or **immediates**.

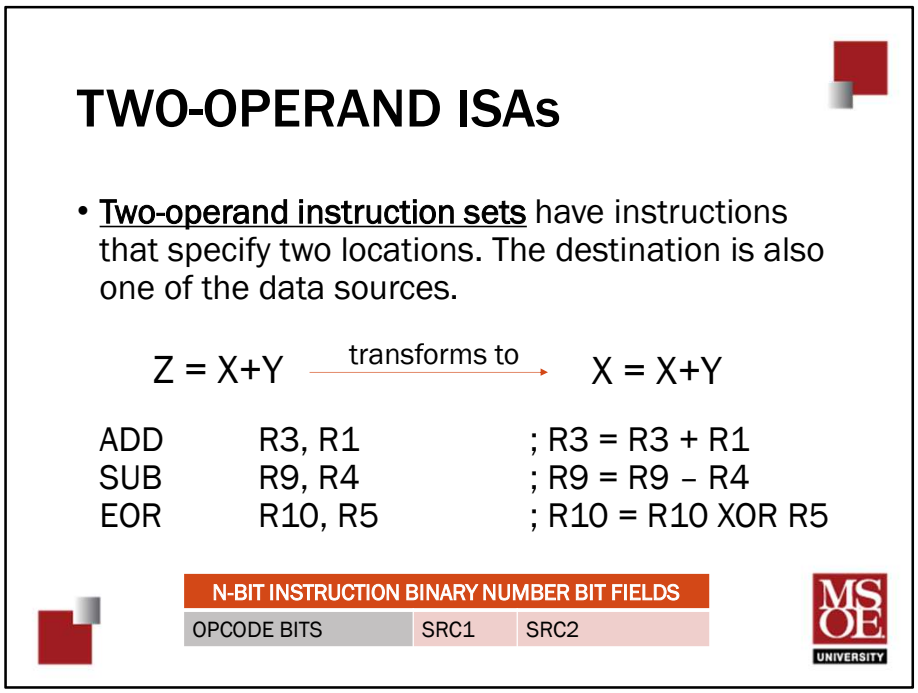


The natural way of writing algebra is using three algebraic variables, and thus three memory locations or three-operands.

- A good example is three-dimensional space. In some computer graphics problem, the z-axis location might be calculated based on the x-axis and y-axis values. Perhaps the equation is z = x + y.
- This example shows the algebra equation converted into a RISC ALU ADD instruction operating on registers. Based on the comments provided behind the semicolons, the first register specified in the instruction is the destination of the calculated result, the second register is source 1 – the one on the left of the arithmetic operation – and the third register is source 2.

- In the instruction binary number, all three memory locations are specified as bit fields.

The order of the operands in the instruction is also specified in the instruction set architecture blueprint. This example has **opcode dest, src1, src2**. Most instruction sets use this format because it is the natural way of writing equations with the destination of the result algebraically on the left of the equals sign. But another instruction set might use **opcode src1, src2, dest**. You must study the blueprint of every instruction set you learn.



In order to shorten the binary number so that programs take less space in instruction memory, some instruction sets require the result to **overwrite** one of the source locations. This type of instruction set is called a two-operand instruction set.

- While two-operand instruction sets might save space in instruction memory, it is not the natural way to write algebra equations.
- Embedded systems often have small program memories to help keep consumer cost low. Thus, instruction set architectures designed for the embedded systems market are often two-operand so that programs can fit in the available memory space.
- Programmers have to adjust the way they think about their mathematics and data movement when working in two-operand instruction sets.

KEY POINTS

- All instructions calculate values, addresses, or move data between memory locations.
- Opcodes identify instructions to the CPU circuit.
- Operands specify data locations.
- Three-operand ISAs specify three data locations.
- Two-operand ISAs specify two data locations.