Floating point numbers and Memory

University of South Carolina

Introduction to Computer Architecture
Fall, 2024
Mehdi Yaghouti

Molinaroli College of Engineering and Computing
UNIVERSITY OF SOUTH CAROLINA

Fixed Point Representation

- Fixed-point notation has an implied binary point

8's column
4's column
2's column
1's column
0.5's column
0.25's column
0.125's column
0.0625's column

$$0110.1011 = 0\times 2^{3} + 1\times 2^{2} + 1\times 2^{1} + 0\times 2^{0} + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} + 1\times 2^{-4} = 6.6875$$

implied

Fixed Point Representation

- Fixed-point notation has an implied binary point

8's column, 4's column, 2's column, 1's column, 0.5's column, 0.25's column, 0.125's column, 0.0625's column

$$0\,1\,1\,0.1\,0\,1\,1 = 0\times2^{3}+1\times2^{2}+1\times2^{1}+0\times2^{0}+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}+1\times2^{-4} = 6.6875$$

implied

- Decimal fraction to Binary conversion
  - Common negative power of two
    0.5, 0.25, 0.125, 0.0625, 0.03125, $\cdots$
  - Repeated multiplication by 2

Fixed Point Representation

- Fixed-point notation has an implied binary point

8's column, 4's column, 2's column, 1's column, 0.5's column, 0.25's column, 0.125's column, 0.0625's column

$$0\,1\,1\,0.1\,0\,1\,1 = 0\times 2^{3} + 1\times 2^{2} + 1\times 2^{1} + 0\times 2^{0} + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} + 1\times 2^{-4} = 6.6875$$

implied

- Decimal fraction to Binary conversion
  - Common negative power of two
    0.5, 0.25, 0.125, 0.0625, 0.03125, $\cdots$
  - Repeated multiplication by 2
- Binary to decimal conversion
  $(b_N \dots b_k \bullet b_{k-1} \dots b_0)_2 = \sum_{i=0}^{N} 2^{i-k} b_{i-k}$

Fixed Point Representation

- Sign/Magnitude

$$0110.1011 \xrightarrow{negate} 1110.1011$$

Signed Fixed Point Representation

- Sign/Magnitude

$$0\,1\,1\,0.1\,0\,1\,1 \xrightarrow{\textit{negate}} 1\,1\,1\,0.1\,0\,1\,1$$

- Two's complement

$$0\,1\,1\,0.1\,0\,1\,1 \xrightarrow{\textit{negate}} 1\,0\,0\,1.0\,1\,0\,1$$

Signed Fixed Point Representation

- Sign/Magnitude

$$0110.1011 \xrightarrow{\textit{negate}} 1110.1011$$

- Two's complement

$$0110.1011 \xrightarrow{\textit{negate}} 1001.0101$$

- `Ua.b` designates an unsigned fixed-point number with a integer and b fraction bits
- `Qa.b` designates an signed fixed-point number with a integer and b fraction bits

Fixed Point Representation

- Compute $1.75 + (-1.625)$ using `Q3.5` fixed-point numbers

Floating Point Representation

- Floating-point numbers are analogous to scientific notation
- In general each rational number can be represented in scientific notation as,

$$\pm d_0.d_1d_2\cdots d_{n-1}d_n \times b^e$$

where b is the base (radix), e is the exponent and each digit $0 \leq d < b$

- As we represent information in binary patterns, the radix is naturally taken as 2
- Example:

$$-765_{10} = -101.1111101_2 \times 2^7$$

- As it can be seen the binary point can *float* at the expense of changing the exponent

Floating Point Representation

- We call a representation normalized if there is only one digit 1 before the radix point
- Example:

$$-765_{10} = -101.1111101_2 \times 2^7$$

$$-765_{10} = -0.1011111101_2 \times 2^{10}$$

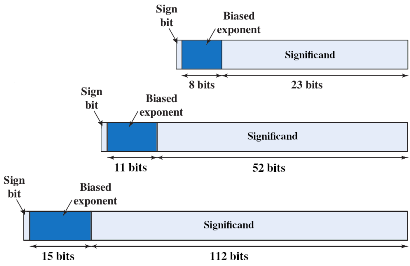
$$-765_{10} = -1.011111101_2 \times 2^9$$

- In computer representation the radix is tacitly assumed to be 2
- The IEEE 754 is the widespread technical standard for floating point representation

Floating Point Representation

- The IEEE 754 standard uses a normalized notation with three fields:
  - **sign**
    - 0 for positive numbers
    - 1 for negative numbers
  - **biased exponent**
    - Represents the sum of the actual exponent and a bias constant
  - **significand (mantissa)**
    - All bits to the right of the binary point
- The 1 to the right of binary point is tacitly assumed for the sake of efficiency

- **Single Precision:** 32 bits
  - Exponent Bias: 127
- **Double Precision:** 64 bits
  - Exponent Bias: 1023
- **Quad Precision:** 128 bits
  - Exponent Bias: 16363



Floating Point Representation

- A floating-point number represented in IEEE 754 format can be calculated as,

$$f = (-1)^S \times (1 + significand) \times 2^{exponent-bias}$$

- Which number is given by the half-precision floating-point `0XC0FF`?

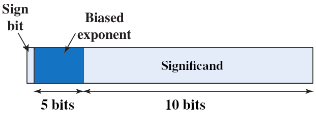
Floating Point Representation

- A floating-point number represented in IEEE 754 format can be calculated as,

$$f = (-1)^S \times (1 + significand) \times 2^{exponent-bias}$$

- Which number is given by the half-precision floating-point `0XC0FF`?

1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1



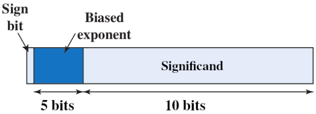
Floating Point Representation

- A floating-point number represented in IEEE 754 format can be calculated as,

$$f = (-1)^S \times (1 + significand) \times 2^{exponent-bias}$$

- Which number is given by the half-precision floating-point `0XC0FF`?

1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1



- sign bit $= 1$
- exponent - bias $= 16 - 15 = 1$
- significand $= \sum_{k=3}^{10} 2^{-k} = 0.249023437$

$$f = (-1)^1 \times (1 + 0.249023437) \times 2^1 = -2.498046874$$

Floating Point Representation

- Encode the number -0.3 into 32-bits single precision format.

Floating Point Representation

- Encode the number -0.3 into 32-bits single precision format.

$$
\begin{aligned}
0.3_{10} &= (0.0100110011001100110011001100110011\cdots)_2 \\
&= (1.00110011001100110011001)_2 \times 2^{-2}
\end{aligned}
$$

Floating Point Representation

- Encode the number -0.3 into 32-bits single precision format.

$$0.3 = (0.0100110011001100110011001100110011\cdots)_2$$
$$= (1.00110011001100110011001)_2 \times 2^{-2}$$

- Sign bit $= 1$
- Biased Exponent $= 127 - 2 = 125 = 01111101_2$
- Significand $= 00110011001100110011001$
- Single Precision Representation $= 1\,01111101\,00110011001100110011001$
- Single Precision Representation $=$ `0XBE999999`
- Absolute error $\approx 1.8 \times 10^{-8}$

Floating Point Representation

- Special Values

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000
QNaN	X	11111111	0 Non-zero
SNaN	X	11111111	1 Non-zero
Sub-Normal	X	00000000	Non-zero

Floating Point Representation

- Rounding Modes
  - **Round to nearest:** The result is rounded to the nearest representable
    - `Overflows` are rounded up to $\pm\infty$
    - `Underflows` are rounded up to 0
  - **Round toward** $+\infty$**:** The result is rounded up toward plus infinity number
  - **Round toward** $-\infty$**:** The result is rounded down toward minus infinity number
  - **Round toward** 0**:** The result is rounded toward zero
- **Overflow :** When the number magnitude is too large to be represented
- **Underflow:** When the number magnitude is too tiny to be represented

Floating Point Addition

- Add two single-precision float numbers `a=0XF2D20004` and `b=0X76407020`.
    1. Extract exponent and significand

Floating Point Addition

- Add two single-precision float numbers `a=0XF2D20004` and `b=0X76407020`.
  1. Extract exponent and significand

$$a = {\color{red}1}\ {\color{blue}11100101}\ 10100100000000000000100$$

$$b = {\color{red}0}\ {\color{blue}11101100}\ 10000000111000000100000$$

Floating Point Addition

- Add two single-precision float numbers a=`0XF2D20004` and b=`0X76407020`.
  1. Extract exponent and significand

$$a = \color{red}{1}\ \color{blue}{11100101}\ 10100100000000000000100$$
$$b = \color{red}{0}\ \color{blue}{11101100}\ 10000000111000000100000$$

  2. Add the leading 1 to significand

$$S_a = \color{green}{1}\ 10100100000000000000100$$

$$S_b = \color{green}{1}\ 10000000111000000100000$$

Floating Point Addition

- Add two single-precision float numbers a=0XF2D20004 and b=0X76407020.
  1. Extract exponent and significand

$$a = 1\ 11100101\ 10100100000000000000100$$
$$b = 0\ 11101100\ 10000000111000000100000$$

  2. Add the leading 1 to significand

$$Sig_a = 1\ 10100100000000000000100$$

$$Sig_b = 1\ 10000000111000000100000$$

  3. Compare the exponents and shift the smaller significand if necessary

$$\begin{array}{r} 11101100 \\ -11100101 \\ \hline 00000111 \end{array}$$

$$Sig'_a = 0000000110100100000000000$$

Floating Point Addition

- Add two single-precision float numbers a=0XF2D20004 and b=0X76407020.
    1. Extract exponent and significand
    2. Add the leading 1 to significand
    3. Compare the exponents and shift the smaller significand if necessary

$$\begin{array}{r} 11101100 \\ -11100101 \\ \hline 00000111 \end{array}$$

$$Sig'_a = 000000011010010000000000$$

    4. Subtract or add significands together

$$\begin{array}{r} 110000000111000000100000 \\ -000000011010010000000000 \\ \hline 101111101100110000100000 \end{array}$$

Floating Point Addition

- Add two single-precision float numbers a=0XF2D20004 and b=0X76407020.
  1. Extract exponent and significand
  2. Add the leading 1 to significand
  3. Compare the exponents and shift the smaller significand if necessary
  4. Subtract or add significands together

$$\begin{array}{r} 110000000111000000100000 \\ -000000011010010000000000 \\ \hline 101111101100110000100000 \end{array}$$

  5. Normalize and readjust the exponent

     $So\ far\ we\ have: \color{red}{0}\ \color{blue}{11101100}\ \color{green}{1}01111101100110000100000$

Floating Point Addition

- Add two single-precision float numbers a=0XF2D20004 and b=0X76407020.
  1. Extract exponent and significand
  2. Add the leading 1 to significand
  3. Compare the exponents and shift the smaller significand if necessary
  4. Subtract or add significands together
  5. Normalize and readjust the exponent
     $So\ far\ we\ have: 0\ 11101100\ 101111101100110000100000$
  6. Rounding the result if needed

Floating Point Addition

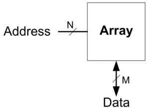
- Add two single-precision float numbers a=0XF2D20004 and b=0X76407020.
  1. Extract exponent and significand
  2. Add the leading 1 to significand
  3. Compare the exponents and shift the smaller significand if necessary
  4. Subtract or add significands together
  5. Normalize and readjust the exponent
     $So\ far\ we\ have$: 0 11101100 101111101100110000100000
  6. Rounding the result if needed
  7. Assemble exponent and fraction back into floating-point format
     $By\ eliminating\ the\ leading\ one$:

$$0\ 11101100\ 01111101100110000100000$$

Therefore: 0XF2D20004 + 0X76407020 = 0X763ECC20

Memory Arrays

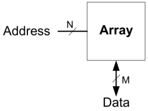
- Registers built from flip-flops, stores small amounts of data
- Memory arrays can efficiently store large amounts of data
- Memories all provides the same generic functionality
- They differ in underlying structures, delay and area
- Memory can be considered as a two dimensional array



- A memory with N-bit address and M-bit data has 2^N rows and M columns

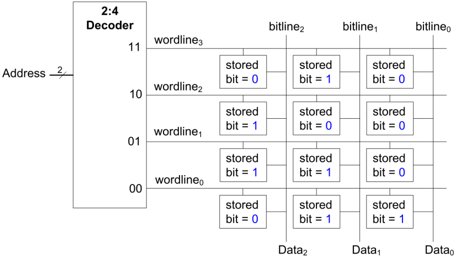
Memory Arrays

- Registers built from flip-flops, stores small amounts of data
- Memory arrays can efficiently store large amounts of data
- Memories all provides the same generic functionality
- They differ in underlying structures, delay and area
- Memory can be considered as a two dimensional array



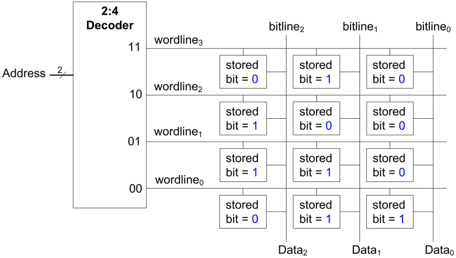
- A memory with N-bit address and M-bit data has 2^N rows and M columns
- The number of rows and columns are called *depth* and *width* of the memory

Memory Arrays



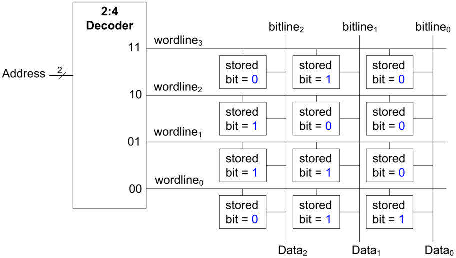
- Memory arrays are built as an array of bit cells
- For each combination of address bits the decoder asserts one wordline
- A bit cell is connected to the bitline, only if its associated wordline is asserted

Memory Arrays



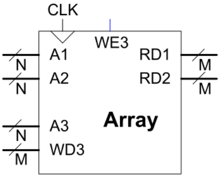
- **Memory read**
  1. Initially the bitline is float (Z)
  2. The wordline gets asserted and connects the bits in the row to the bitlines

Memory Arrays



- **Memory Write**
  1. The bitlines are strongly driven to 1 or 0 depending on the data
  2. The wordline gets asserted and connects the bits in the bitlines
  3. The bitlines overpower the content of each bit cell and overwrite the new value

Memory Arrays



- As a black box, regardless of type, each memory is a multi-ported array of bits
- A Multiported memory can access several addresses simultaneously
- Writing into a memory address happens on the edge of the clock

Memory Arrays

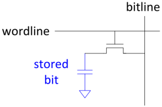
- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
  - **ROM:** is non-volatile

Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are stored in inter-coupled inverters
  - **ROM:** is non-volatile

Memory Arrays

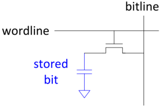
- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - The bit value will be destroyed by reading or charge leakage
    - The contents must be refreshed every few milliseconds



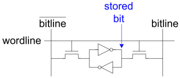
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile

Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - The bit value will be destroyed by reading or charge leakage
    - The contents must be refreshed every few milliseconds



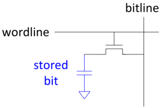
    - **SRAM**: bits are interlocked in cross-coupled inverters
    - Bit values do not need to be refreshed
    - Cross-coupling of inverters robustify them against noise



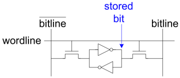
  - **ROM:** is non-volatile

Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - The bit value will be destroyed by reading or charge leakage
    - The contents must be refreshed every few milliseconds



    - **SRAM**: bits are interlocked in cross-coupled inverters
    - Bit values do not need to be refreshed
    - Cross-coupling of inverters robustify them against noise
    - Trade-off between area and speed

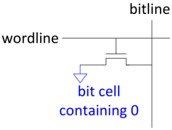


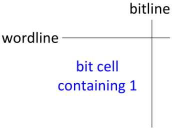
Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow

  - **ROM:** is non-volatile

Memory Arrays

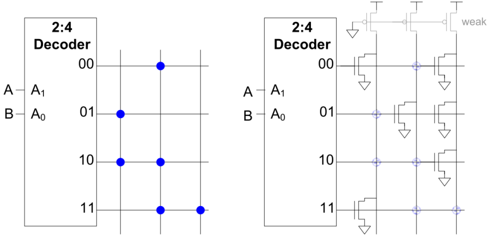
- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile
    - bits can be stored as presence or absence of transistors
    - bitlines are driven weakly high
    - transistors with asserted wordline pull down the value to zero





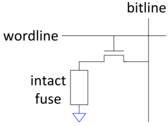
Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile
    - presence or absence of transistors can be depicted as dots

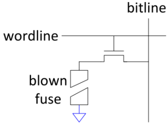


Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile
    - **PROM:** Programmable ROM



bit cell containing 0



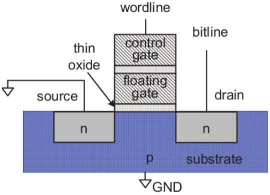
bit cell containing 1

Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile
    - **PROM:** Programmable ROM
    - **EPROM:** Erasable Programmable ROM
    - Use floating gate transistors
    - Need high voltage to be programmed
    - Need intense UV to be erased

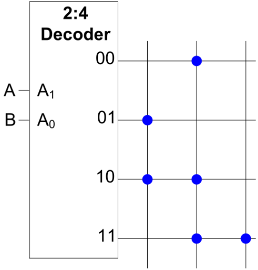
Memory Arrays

- Memories differ in how they store bits in bit cells
- **Memory types**
  - **RAM:** is volatile
    - **DRAM**: bits are stored as charge on capacitors
    - **SRAM**: bits are interlocked in cross-coupled inverters
  - **ROM:** is non-volatile
    - **PROM:** Programmable ROM
    - **EPROM:** Erasable Programmable ROM
    - **EEPROM:** Electrically Erasable Programmable ROM



Look Up Table (LUT)

- Memory arrays can also be used to perform combinational logic functions
- It looks up outputs for input combinations, matching addresses to truth table rows
- Memory arrays used for logic functions are called lookup tables (LUTs)
- A 2^N-word $\times$ M-bit memory can perform any combinational function of N inputs and M outputs



SystemVerilog

- The following modules describes a $2^N \times M - bit$ RAM
- Writes occur at the clock edge only if the `we` is asserted
- Reads occur in a combinational manner

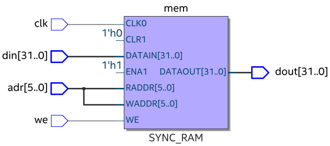
```
module ram #(parameter N = 6, M = 32) ( input  logic clk,
                                        input  logic  we,
                                        input  logic [N-1:0] adr,
                                        input  logic [M-1:0] din,
                                        output logic [M-1:0] dout);

    logic [M-1:0] mem [2**N-1:0];

    always_ff @(posedge clk)
      if (we) mem [adr] <= din;

    assign dout = mem[adr];

endmodule
```



SystemVerilog

- The following modules describes a $2 \times 3 - bit$ ROM
- Reads occur with in a combinational manner

```
module rom( input  logic [1:0] adr,
            output logic [2:0] dout)

    always_comb
     case(adr)
        2'b00: dout <= 3'b011;
        2'b01: dout <= 3'b110;
        2'b10: dout <= 3'b100;
        2'b11: dout <= 3'b010;
     endcase
endmodule
```