Read Section 4 of Std 1364-2001, write in the answer at A: for each of the following questions, and bring to class on Thursday, February 6.

**Operators**

Clearly describe the function that each of the operators listed performs:

- **Arithmetic (binary)**
  - `+`: addition
  - `-`: subtraction
  - `*`: multiplication
  - `/`: division
  - `%`: modulus

- **Arithmetic (unary)**
  - `+`: + sign for number
  - `-`: - sign for number

- **Bitwise**
  - `~`: Bitwise negation (NOT)
  - `&`: Bitwise AND
  - `|`: Bitwise OR
  - `^`: Bitwise XOR
  - `~^` or `^~`: Bitwise XNOR

- **Reduction**
  - `&`: Reduction AND
  - `~&`: Reduction NAND
  - `|`: Reduction OR
  - `~|`: Reduction NOR
  - `^`: Reduction XOR
  - `~^` or `^~`: Reduction XNOR

- **Logical**
  - `!`: Logical Negation
  - `&&`: Logical AND
  - `||`: Logical OR
  - `==`: Logical Equality
  - `!=`: Logical Inequality

- **Relational**
  - `<`: Less Than
  - `<=`: Greater Than or Equal to
  - `>`: Greater Than
  - `>=`: Less Than or Equal to

**Expression Bit Widths**

Depends on:
- widths of operands and types of operators
- Verilog fills in smaller-width operands by using zero extension.
- Final or intermediate result width may increase expression width

Width (Unsized constant number) = same as integer (usually 32)
Width (Sized constant number) = number of bits specified.

**Arithmetic binary and bitwise**

\[ x \text{ op } y \text{ where op is } +, -, *, /, \%, &, |, \land, \lor, ^, ~\land, ~\lor, ~^ \]

Bit width = max(width(x), width(y))

What is the width of `A + B` for `reg [7:0] A; reg [15:0] B;`?

\[ A: 16 \text{ bits} \]

Will the assignment statement `assign result = (A+B) >> 1;` work properly for all possible pairs of 16-bit operands for `reg [15:0] A, B, result;`?

\[ A: No. The carry out of the addition is lost before the shift occurs even though it could have been recovered with a width of 17. Make one or both of the A and B operands one bit longer. \]

**Arithmetic unary**

op x where op is +, -

Bit width = width(x)

**Bitwise negation**

op x where op is ~

Bit width = width(x)

**Logical, relational and reduction**

\[ x \text{ op } y \text{ where op is } ==, !=, <=, >=, &&, ||, ?>, <<=, <=, or op y \text{ where op is } !, \lor, \land, ~\land, ~\lor, ~^ \]

Bit width = 1

What is the bit width of `(X & (A + B))` for `wire X; wire [7:0] A, B;`?

\[ A: 1 \text{ bit} \]
Shift
\( \text{op} \ y \) where op is <<, >>
- Bit width = width(x)

Conditional
\( x ? y : z \)
- Bit width = max(width(y), width(z))

What is the bit width of \( \text{assign result} = X ? A:B \) for wire X; \( \text{reg}[9:0] \) A; \( \text{reg}[18:10] \) B; A: 10 bits

Concatenation
\( \{x, ..., y\} \)
- Bit width = width(x) + ... + width(y)

What is the bit width of \( \{A, B, C\} \) for wire A; \( \text{reg}[7:0] \) B; \( \text{reg}[15:0] \) C; A: 25 bits

Replication
\( \{x, y, ..., z\} \)
- Bit width = \( x \times (\text{width(y)} + ... + \text{width(z)}) \)

Expressions with Operands Containing x or z

Arithmetic
If any bit is x or z, result is all x's.
Divide by 0 produces all x's.

What is the result of \( \text{assign result} = A + B \); for A = 4'b1100, B = 2'b11?
A: 1111

What is the result of \( \text{assign result} = A + B \); for A = 4'b1100, B = 4'b0111?
A: xxxx

Relational
If any bit is x or z, result is x.

Logical
\( = = \) and \( != \): If any bit is x or z, result is x.
\( = = = \) and \( != = \): All bits including x and z values must match for equality;
otherwise unequal. Lengths of operands must be equal.
Operands are reduced to bit 0 or 1, respectively, based on zero or nonzero value
What is the value of \( \text{4'b0011} = = \text{2'b11} \)?
A: 1, the right operand is zero-filled to 4 bits and they are then equal.
Can \( \text{4'b0011} = = \text{2'b11} \) be evaluated? Explain your answer.
A: Yes, since the operands are of unequal length, the result is 0.
\&\&, || combines logical values 0, 1, x according to:

Bitwise
Defined by tables for 0, 1, x, z operands.

Reduction
Defined by tables as for bitwise operators.

Shifts
z changed to x.
Vacated positions zero filled.