Lecture 7: Floating Point

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

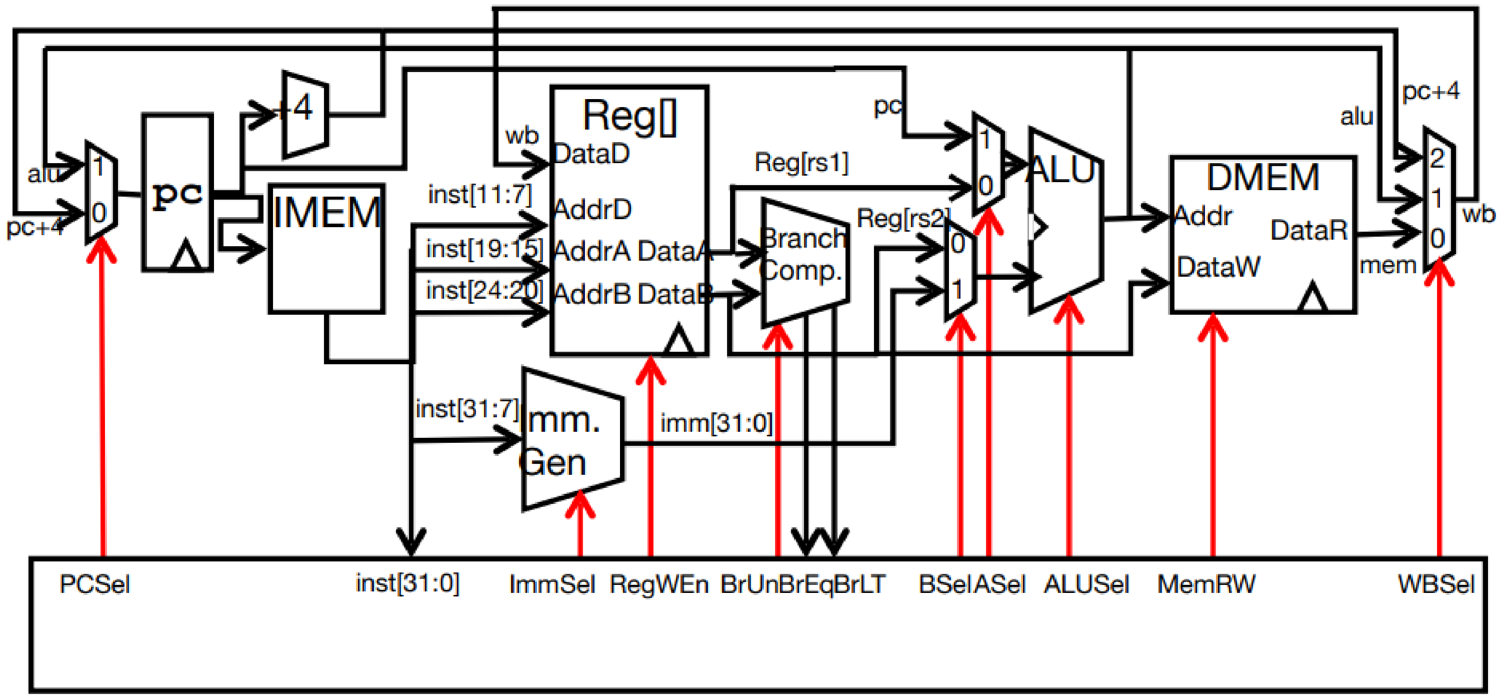
- Slides were developed in the reference with
  - CS 61C at UC Berkeley
    - https://inst.eecs.berkeley.edu/~cs61c/sp23/
  - CS 252 at UC Berkeley
    - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
  - CSCE 513 at University of South Carolina
    - https://passlab.github.io/CSCE513/

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

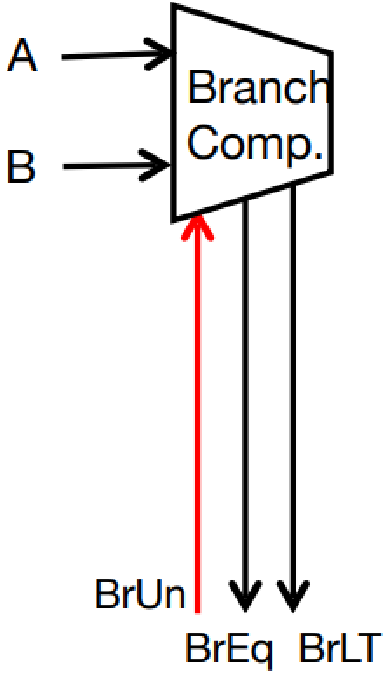
Datapath with Branches(1/2)

- **Datapath with Branches**



Datapath with Branches(2/2)

- **Branch Comparator**

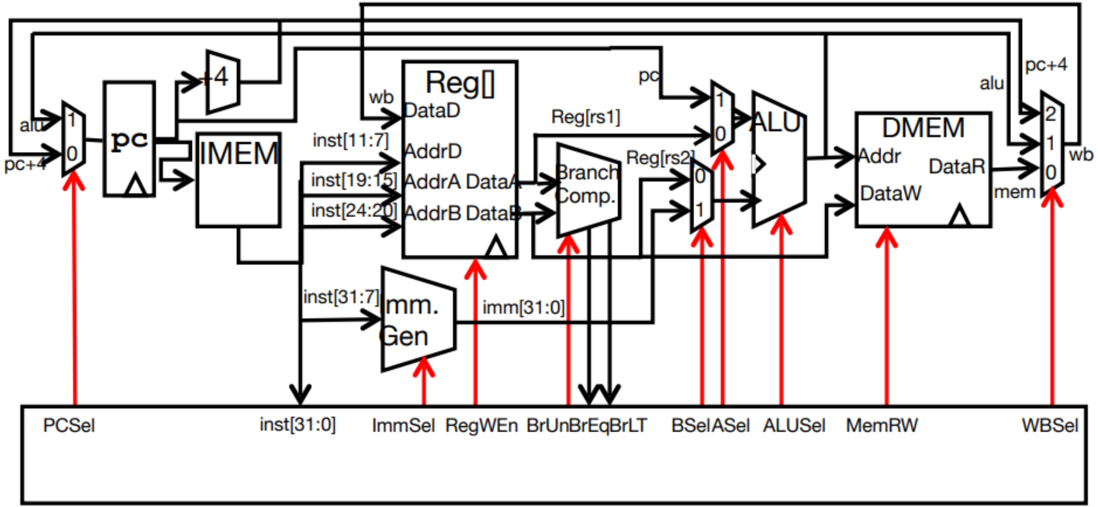


- BrEq = 1, if A=B
- BrLT = 1, if A < B
- BrUn =1 selects unsigned comparison for BrLT, 0=signed
- BGE branch: A >= B, if !(A<B)

Takeaway Questions

- What are proper control signals for **lui** instruction?
  - (a) Bsel = 0, Asel = 0, WBSel = 0
  - (b) Bsel = 0, Asel = 1, WBSel = 1
  - (c) Bsel = 1, Asel = 1, WBSel = 1

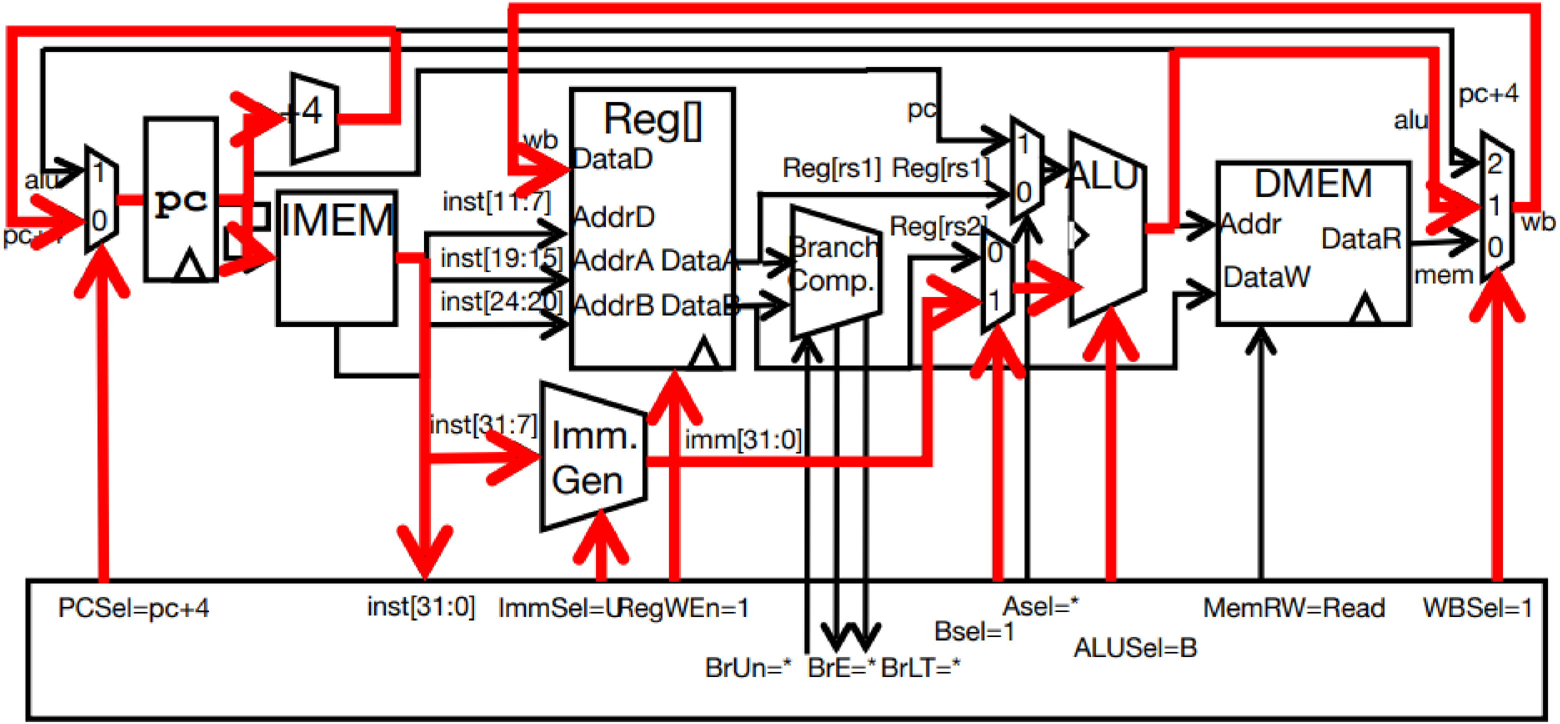
lui rd, uimm20



Takeaway Questions

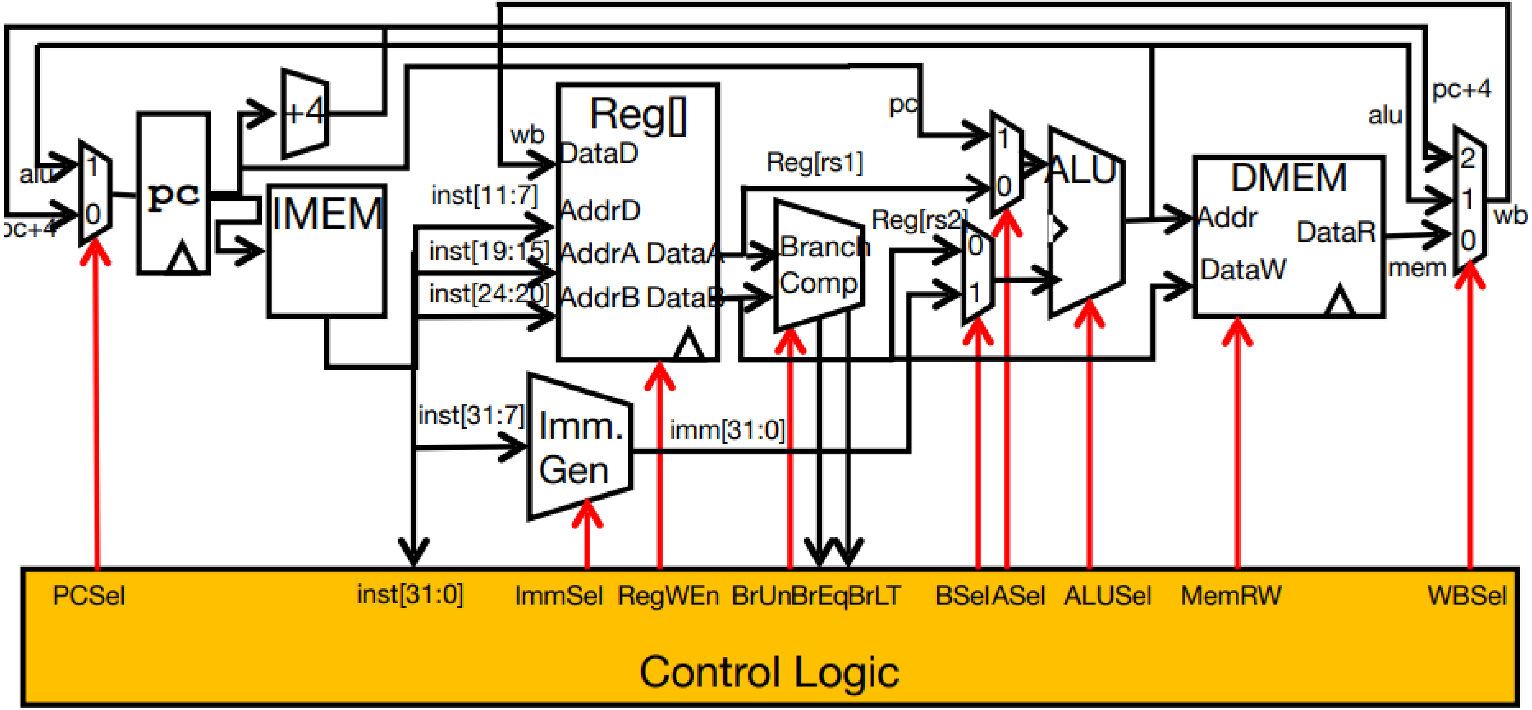
lui rd, uimm20

- What are proper control signals for **lui** instruction?



RV32I Control Logics (1/2)

- **Control Logics**



RV32I Control Logics (2/2)

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

Control Block Design (1/6)

- **ROM**
  - Read-only memory
  - Can be easily reprogrammed
    - Fix errors
    - Add instructions
  - Popular when designing control logic manually
- **Combinatorial Logic**
  - Chip designers use logic synthesis tools to convert truth tables to networks of gates

Control Block Design (2/6)

- Instruction type encoded using only 9 bits

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

inst[30] inst[14:12] inst[6:2]

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

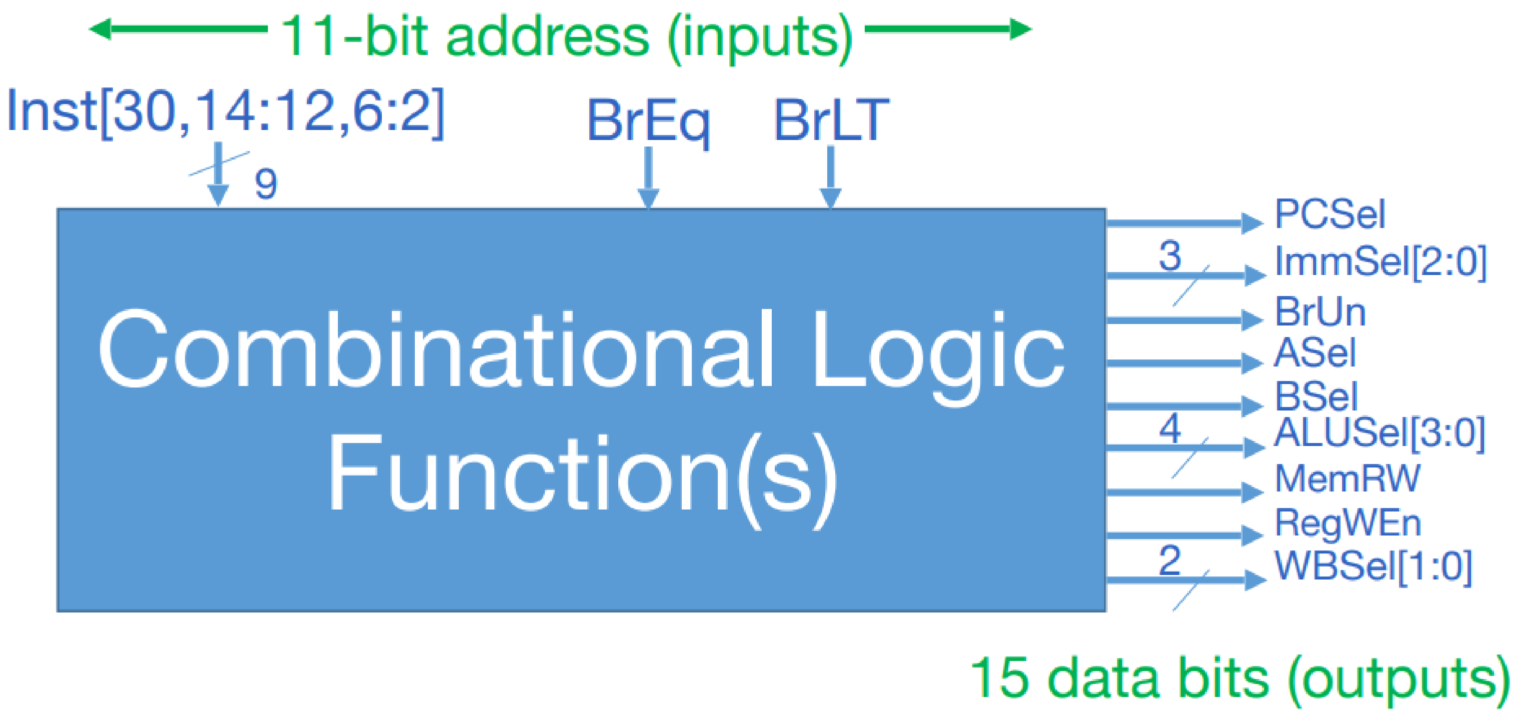
Control Block Design (3/6)

- How to decide whether BrUn is 1?
  - Inst[13] and branch

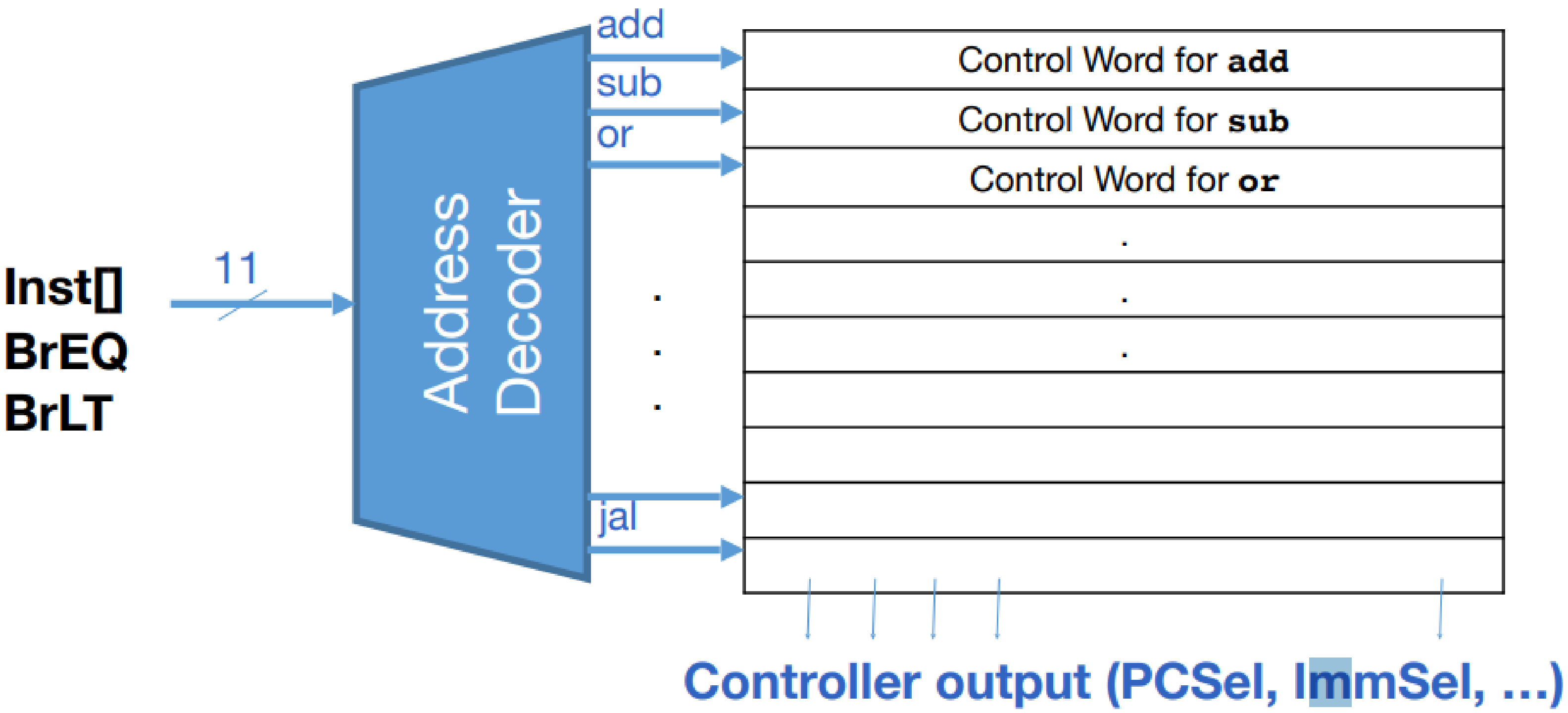
Inst[14:12] Inst[6:2]

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

Control Block Design (4/6)



Control Block Design (5/6)



Control Block Design (6/6)

- How to decode add?

  add = i[30]•i[14]•i[13]•i[12]•R-type

  R-type = i[6]•i[5]•i[4]•i[3]•i[2]•RV32I

  RV32I = i[1]•i[0]

				Inst[14:12]			Inst[6:2]		
0000000	rs2	rs1		000		rd	0110011		BEQ
0100000	rs2	rs1		000		rd	0110011		BNE
0100000	rs2	rs1		001		rd	0110011		BLT
0000000	rs2	rs1		010		rd	0110011		BGE

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

Review of Numbers (1/9)

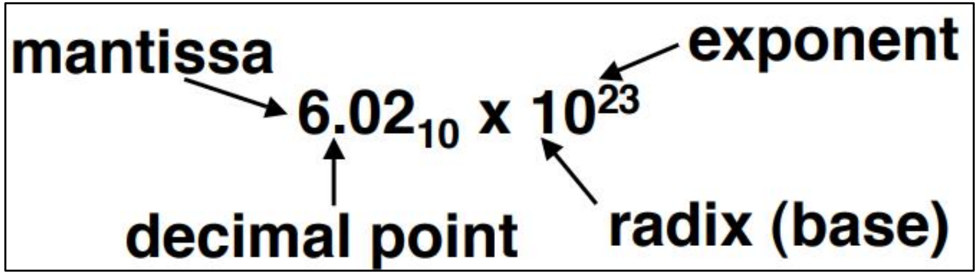
- Computers are made to deal with numbers
- What can we represent in N bits?
  - Unsigned integers
    - 0 to 2^{N-1}
  - Signed Integers (Two's Complement)
    - $-2^{(N-1)}$ to $2^{(N-1)} - 1$

Review of Numbers (2/9)

- **What about other numbers?**
  - Very large numbers?
    - $3{,}155{,}760{,}000_{10}$ ($3.15576_{10} \times 10^9$)
  - Very small numbers? (atomic diameters)
    - 0.00000001_{10} ($1.0_{10} \times 10^{-8}$)
  - Rationals (repeating pattern)
    - 2/3 (0.666666666…)
  - Irrationals
    - $2^{1/2}$ (1.414213562373….)
  - Transcendentals
    - e(2.718…), pi(3.141…)

Review of Numbers (3/9)

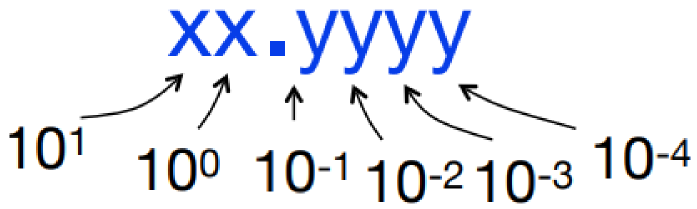
- **Scientific Notation (in Decimal)**



  - Normalized form: no leading 0s
    - (exactly one digit to the left of the decimal point)
  - Alternatives to representing 1/1,000,000,000
    - Normalized: 1.0×10^{-9}
    - Not normalized: 0.1×10^{-8}; 10.0×10^{-10}

Review of Numbers (4/9)

- **Representation of fractions (in Decimal)**
- **Example 6-digit representation**



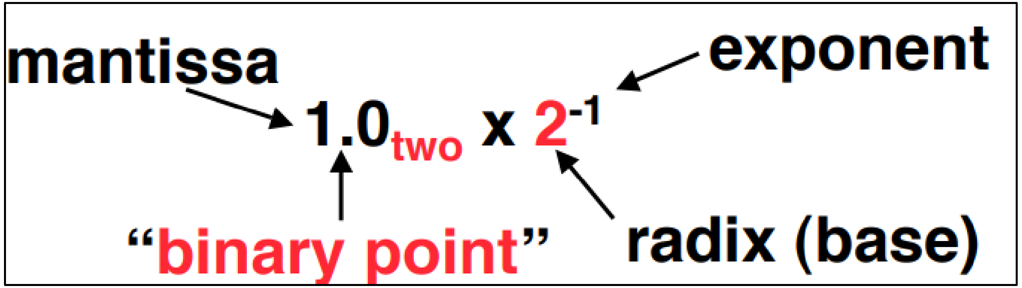
  - $25.2406_{10} = 2\text{x}10^{1} + 5\text{x}10^{0} + 2\text{x}10^{-1} + 4 \text{ x } 10^{-2} + 6\text{x}10^{-4}$

Review of Numbers (5/9)

- **Representation of fractions (in Decimal)**
- **How to store 6.022×10^{23}**
  - Sign bits: 1 bit (+/-)
  - Mantissa: 4 decimal digits (6022)
    - Positive integer with no leading zeros
    - We can save it as an unsigned number
  - Exponent: 23
    - Positive or negative integer

Review of Numbers (6/9)

- **Scientific Notation (in Binary)**



  - Computer arithmetic that supports it is called <u>floating point</u>
  - Floating point represents numbers where the binary point is not fixed, as it is for integers
    - Declare such variable in C as float

Review of Numbers (7/9)

- **Representation of fractions (in Binary)**
- **Example 6-digit representation**

xx.yyyy

2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

  - $10.1010_2 = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{10}$
  - If we assume a “fixed binary point”, the range of 6-bit representations with this format: is 0 to 3.9375

Review of Numbers (8/9)

- **Addition in the representation of the fraction**

$$
\begin{array}{rr}
01.100 & 1.5_{ten} \\
+\ 00.100 & 0.5_{ten} \\
\hline
10.000 & 2.0_{ten}
\end{array}
$$

Review of Numbers (9/9)

- **Multiplication in the representation of the fraction**
- **Where is the answer?**
  - 0.11_2 ($0.5_{10} + 0.25_{10} = 0.75_{10}$)
  - Need to remember where point is

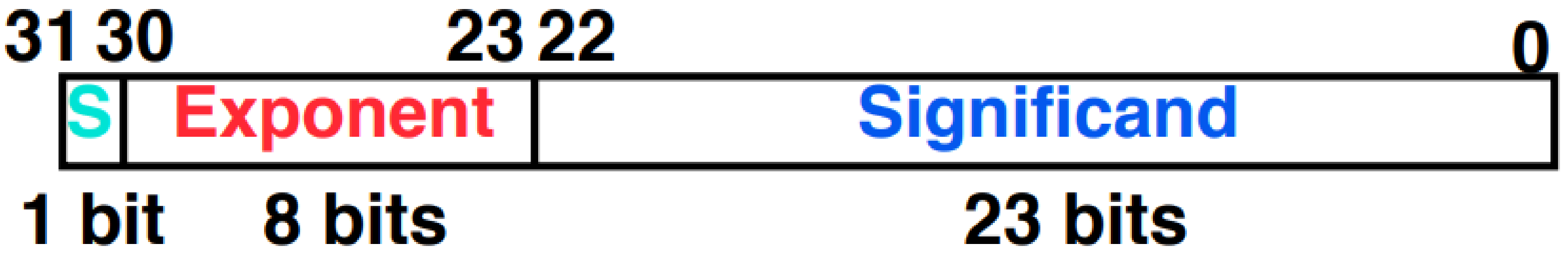
```
       01.100   1.5
       00.100   0.5
      --------
       00 000
      000 00
    0 110 0
   00 000
  000 00
 -----------
0000 110000
0000.110000
```

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

IEEE 754 single precision Floating-Point (1/8)

- **Normal format: +1.xxxxxxxxxx$_{two}$ *2^{yyyyy}$_{two}$**
- Multiple of word size (32 bits)



  - S represents Sign
  - Exponent represents y's
  - Significand represents x's
  - Represent numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}

IEEE 754 single precision Floating-Point (2/8)

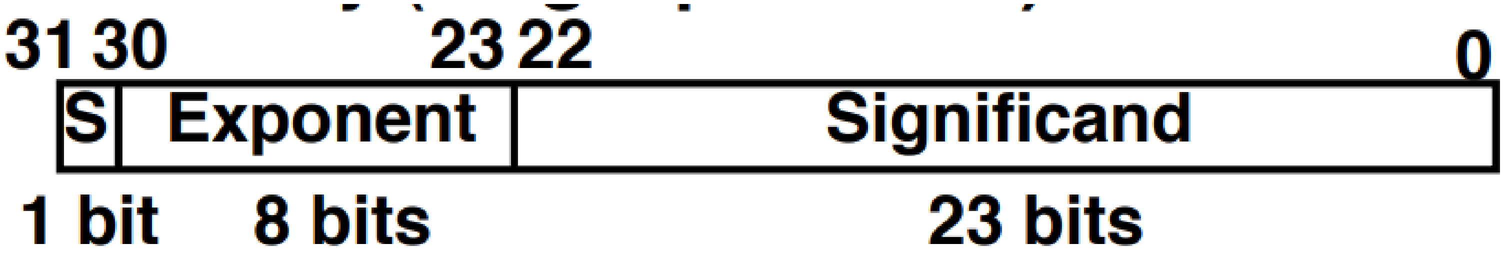
- **For “single precision”, a 32-bit word**
  - 1 bit for **sign(s)** of floating point number
  - 8 bits for **exponent (E)**
  - 23 bits for **fraction (F)**
  - Get 1 extra bit of precision be $(-1)^s \times (1 + F) \times 2^E$ there should always be a 1, so why store it at all?
  - Can represent approximately numbers in the range of 2.0×10^{-38} to 2.0×10^{38}

IEEE 754 single precision Floating-Point (3/8)

- **A negative floating point number?**
  - 2's complement $1000_2 = -8_{10}$
  - $1000_2 = 8_{10}$
  - **Biased notation** stores a signed number N as an unsigned value N+B, where B is the bias.
    - IEEE 754, single precision bias value is 127
    - 2's complement $1111\ 1111_2 = -1_{10}$
      - Bias notation = -1 + 127 = 126 = $0111\ 1110_2$
    - 2's complement $0000\ 0001_2 = 1$
      - Bias notation = 1 + 127 = 128 = $1000\ 0000_2$

IEEE 754 single precision Floating-Point (4/8)

- **Bias Notation**
  - IEEE 754 uses bias of 127 for single precision
  - Subtract 127 from Exponent field to get actual value for the exponent
  - 1023 is bias for double precision



- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

IEEE 754 single precision Floating-Point (5/8)

- **A floating point number uses**
  - x bits for exponent
  - y bits for mantissa
  - Assume a system with 3 exponent bits (bias of -3) and 4 mantissa
    - S XXX MMMM
  - This represents the number $(-1)^{S} * 0bM.MMM * 2^{(0bXXX+(-3))}$

IEEE 754 single precision Floating-Point (6/8)

- **How to convert 10.875 to IEEE 745 FP Format?**
  - Step 1: Write the number in binary
    - $1010.111_2 = 1.010111000\ldots * 2^3$
  - Step 2: Determine Sign/Exponent/Mantissa
    - Sign = Positive ->0
    - Exponent: 3-(-127) = 130 -> $1000\ 0010_2$
    - Mantissa: 1010 1110 0000 0000 0000 0000
  - Step 3: Concatenate
    - 0100 0001 0101 0111 0000 0000 0000 0000

S	Exponent	Mantissa
0	100 0001 0	101 0111 0000 0000 0000 0000

IEEE 754 single precision Floating-Point (7/8)

- **How to convert 0xC3CC0000 to decimal?**
  - Step 1: Write the number in binary
    - 1100 0011 1100 1100 0000 0000 0000 0000
    - C 3 C C 0 0 0 0
  - Step 2: Determine Sign/Exponent/Mantissa
    - Sign = Negative ->1
    - Exponent: $1000\ 0111_2$ -> 135-(127) = 8
    - Mantissa: 1001 1000 … -> 1.001 1000 = $1 + 2^{-3} + 2^{-4}$
    - $(1 + 2^{-3} + 2^{-4}) * 2^8 = 2^8 + 2^5 + 2^4 = 304$

IEEE 754 single precision Floating-Point (8/8)

- **How to convert 0x00000000 to decimal?**
  - Step 1: Determine Sign/Exponent/Mantissa
    - Sign = Positive ->0
    - Exponent: $0000\ 0000_2$ -> 0-(127) = -127
    - Mantissa: 0000 0000 … -> 0
    - $0 * 2^{-127} = 0$

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

The implicit 1 (1/7)

- Our mantissa is guaranteed not to have any leading zeros
  - If we wanted to write $0.234*10^5$, we'd instead write it as $2.34*10^4$
- In binary, every digit is only either 1 or 0
  - Since the MSB can't be 0, it must therefore be 1
  - **If the first bit will always be 1, we don't need to store it!**
  - We can save 1 bit (or alternatively add another bit of precision) to the mantissa by not including the MSB of the mantissa
    - This is known as the implicit 1
    - The resulting mantissa is a "normalized" number

The implicit 1 (2/7)

- **How to convert 10.875 to IEEE 745 FP Format?**
  - Step 1: Write the number in binary
    - $1010.111_2 = 1.\boxed{010111000}\ldots * 2^3$
  - Step 2: Determine Sign/Exponent/Mantissa
    - Sign = Positive ->0
    - Exponent: 3-(-127) = 130 -> $1000\ 0010_2$
    - Mantissa: 0101 1100 0000 0000 0000 000
  - Step 3: Concatenate
    - 0100 0001 0010 1110 0000 0000 0000 0000

S **Exponent** **Mantissa**

The implicit 1(3/7)

- **How to convert 0x00000000 to decimal?**
  - Step 1: Determine Sign/Exponent/Mantissa
    - Sign = 0 -> Positive
    - Exponent: $0000\ 0000_2$ -> 0-(127) = -127
    - Mantissa: 0000 0000 … -> 1.000….
    - $1 * 2^{-127} = 0$

The implicit 1 (4/7)

- **How to convert 0x00000001 to decimal?**
  - Step 1: Determine Sign/Exponent/Mantissa
    - Sign = 0 -> Positive
    - Exponent: $0000\ 0000_2$ -> $0-(127) = -127$
    - Mantissa: 0000 0000 …1 -> 1.000….1 = $1 + 2^{-23}$
    - $(1 + 2^{-23}) * 2^{-127} = 2^{-127} + 2^{-150}$

The implicit 1(5/7)

- **Problems with the implicit 1: Underflow**
  - The smallest number (in absolute value) we can represent is 2^{-127}
  - The underflow: the result of computation gets too small to be represented
- **Solution: Denormalized numbers**
  - If the exponent bits are all zero, then it instead represent
    - $(-1)^S * 0.MMMM * 2^{(000+(-3)+1)}$
  - Ends up losing precision at small numbers (so there is still underflow), but at least it's not a sudden cliff drop

The implicit 1(6/7)

- **How to convert 0x00000000 to decimal?**
  - Step 1: Determine Sign/Exponent/Mantissa
    - Sign = 0 -> Positive
    - Exponent: $0000\ 0000_2$ -> all zeros, so exponent meaning is 0-(127)+1 = -126
    - Mantissa: 0000 0000 … -> 0.000…
    - $0 * 2^{-126} = 0$ (We can represent 0)

The implicit 1(7/7)

- **How to convert 0x00000001 to decimal?**
  - Step 1: Determine Sign/Exponent/Mantissa
    - Sign = 0 -> Positive
    - Exponent: $0000\ 0000_2$ -> all zeros, so exponent meaning is 0-(127)+1 = -126
    - Mantissa: 0000 0000 …1 -> 0.000…1 = 0 + 2^{-23}
    - $(0+2^{-23}) * 2^{-126} = 2^{-149}$ (Much closer to 0)

Outline

- Datapath with Branch
- Control Block Design
- Review of Numbers
- IEEE 754 Format
- The implicit 1
- Exception Handling

Exception Handling (1/6)

- **Exception cases:**
  - We should deal with “1 divided by 0”
  - Ideally, we include an “infinity” value to handle these cases
- **If the exponent bits are all ones, then**
  - If the mantissa is all zeros, it either ∞ or $-\infty$ (depending on sign bit)
  - If the mantissa isn’t all zeros, then it’s a NaN (Not a Number)

Exception Handling (2/6)

- **IEEE-754 standard**
  - SXXX XXXX XMMM MMMM MMMM MMMM MMMM MMMM
- **If the exponent bits are nonzero and not all ones, then**
  - $(-1)^S * 1.MMMM\ldots * 2^{(XXXX\ XXXX+(-127))}$
- **If the exponent bits are all zero, then**
  - $(-1)^S * 0.MMMM\ldots * 2^{(0+(-127)+1)}$
- **If the exponent bits are all ones, then**
  - If the mantissa is all zeros, it either ∞ or -∞ (depending on sign bit)
  - If the mantissa isn’t all zeros, then it’s a NaN (Not a Number)

Exception Handling (3/6)

- **Convert 0xFF80 0000 as an IEEE-754 float to decimal**
  - Step 1: Convert 0xFF80 0000 as binary
    - 1111 1111 1000 0000 0000 0000 0000 0000
    - Sign: 1-> Negative
    - Exponent: 1111 1111. All ones, so we’re dealing with a special case
    - Mantissa: 0000…. -> All zeros
    - -∞

Exception Handling (4/6)

- **Convert 0xFF80 0001 as an IEEE-754 float to decimal**
    - Step 1: Convert 0xFF80 0001 as binary
        - 1111 1111 1000 0000 0000 0000 0000 0000
        - Sign: 1-> Negative
        - Exponent: 1111 1111. All ones, so we’re dealing with a special case
        - Mantissa: 0000…. 1-> Not all zeros
        - NaN

Exception Handling (5/6)

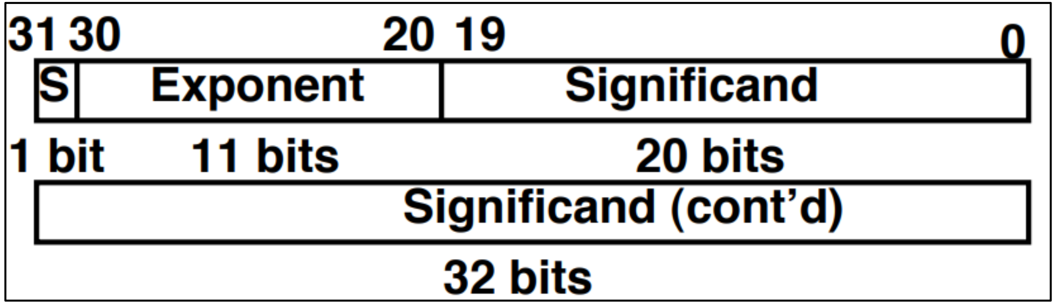
- **Invalid operation**
  - Ex. Sqrt(-1.0)
  - By default, returns a NaN (quiet)
- **Division by zero**
  - By default, return infinity
- **Overflow**
  - By default, return infinity
- **Underflow**
  - Return a denorm. Note that we consider any result using a denorm to suffer from underflow (due to precision loss)
- **Inexact value**
  - Any math that yields a number that can't be exactly represented, like 1.0/3
  - Rounds to a representable number according to the rounding rule

Exception Handling (6/6)

- **IEEE 754: Rounding Rules**
  - The most common is “round to the nearest value”, and break ties to the even number (last bit 0)
    - To round 14.5 to the nearest 2, go to 14, since 14 is closer to 14.5 than 16
    - To round 15 to the nearest 2, go to 16, since 16 ends in more 0 bits than 14

Double Precision Representation

- **Next Multiple of word size (64 bits)**



- **Double Precision (vs. Single Precision)**
  - C variable declared as double
  - Represent numbers almost as small as 2.0×10^{-308} to almost as large as 2.0×10^{308}
  - Primary advantage is greater accuracy due to larger significand

Conclusion

- Floating point: we break up the bucket-o-bits differently
  - A single sign bit (0 == positive, 1 == negative)
  - An exponent in biased form
  - A significant with an implicit leading 1
- Complications occur at the edge
  - Maximum exponent -> Either ∞ or NaN
  - Minimum exponent -> Either 0 or a denormalization
    - Fixed exponent, no more implicit leading 1