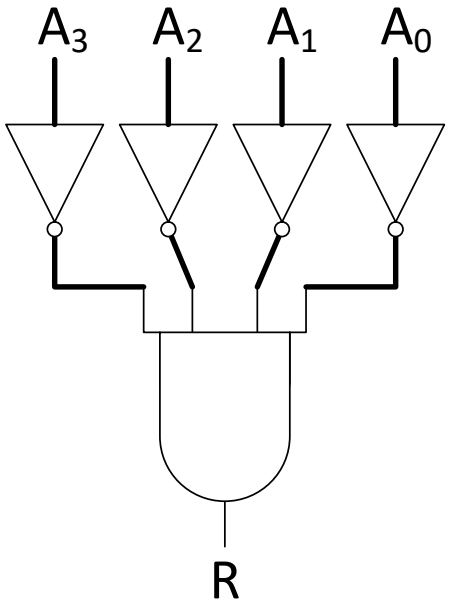
Comparators

Prof. James L. Frankel
Harvard University

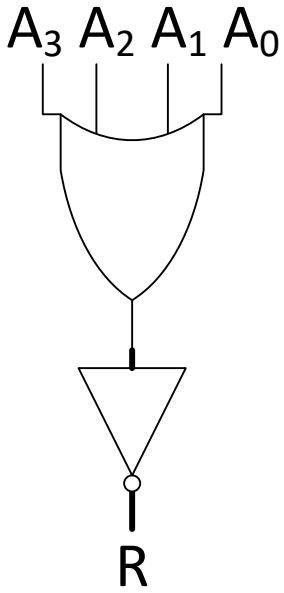
Version of 5:49 PM 12-Nov-2024

Circuit to Determine If a 4-bit Value is Zero

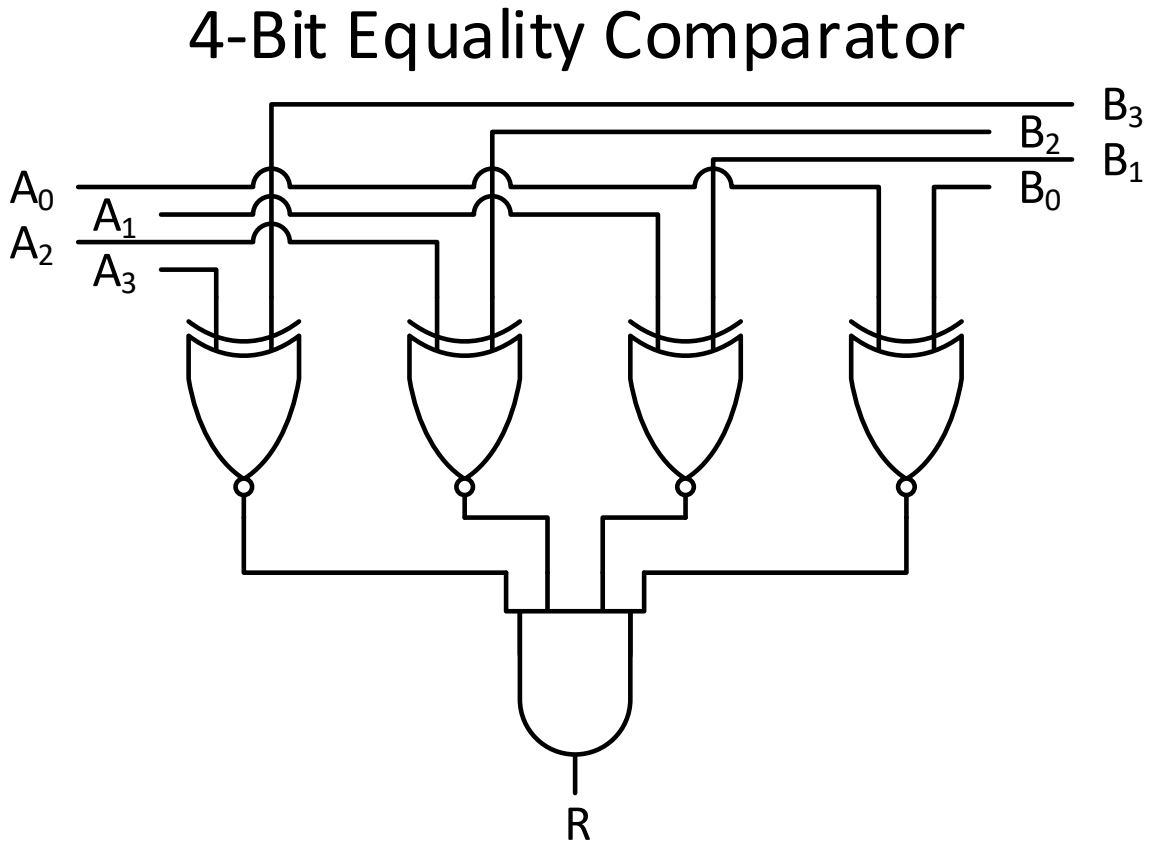
4-Bit Zero Detector



4-Bit Zero Detector



Circuit to Compare Two 4-bit Values for Equality



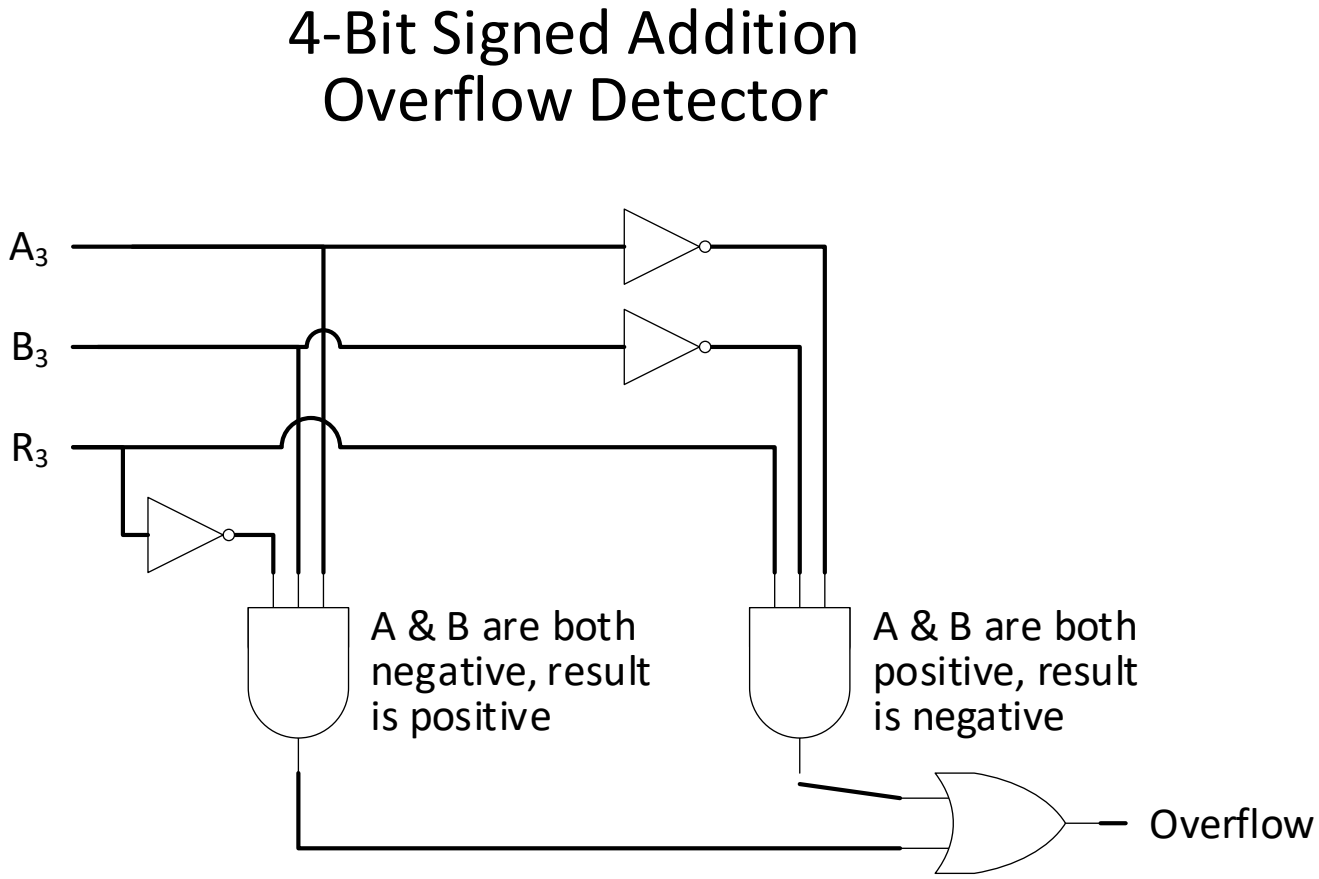
Signed Overflow from an Adder

- Signed overflow occurs from an adder while performing addition when
  - The sign of the result does not have the same sign as the operands

    Or, in other words:
  - The operands are both positive, but the result is negative; or
  - The operands are both negative, but the result is positive

- Signed addition cannot overflow when
  - The operands are of opposite signs

Circuit to Detect Overflow from a 4-bit Adder



Signed Overflow from an Adder when performing Subtraction

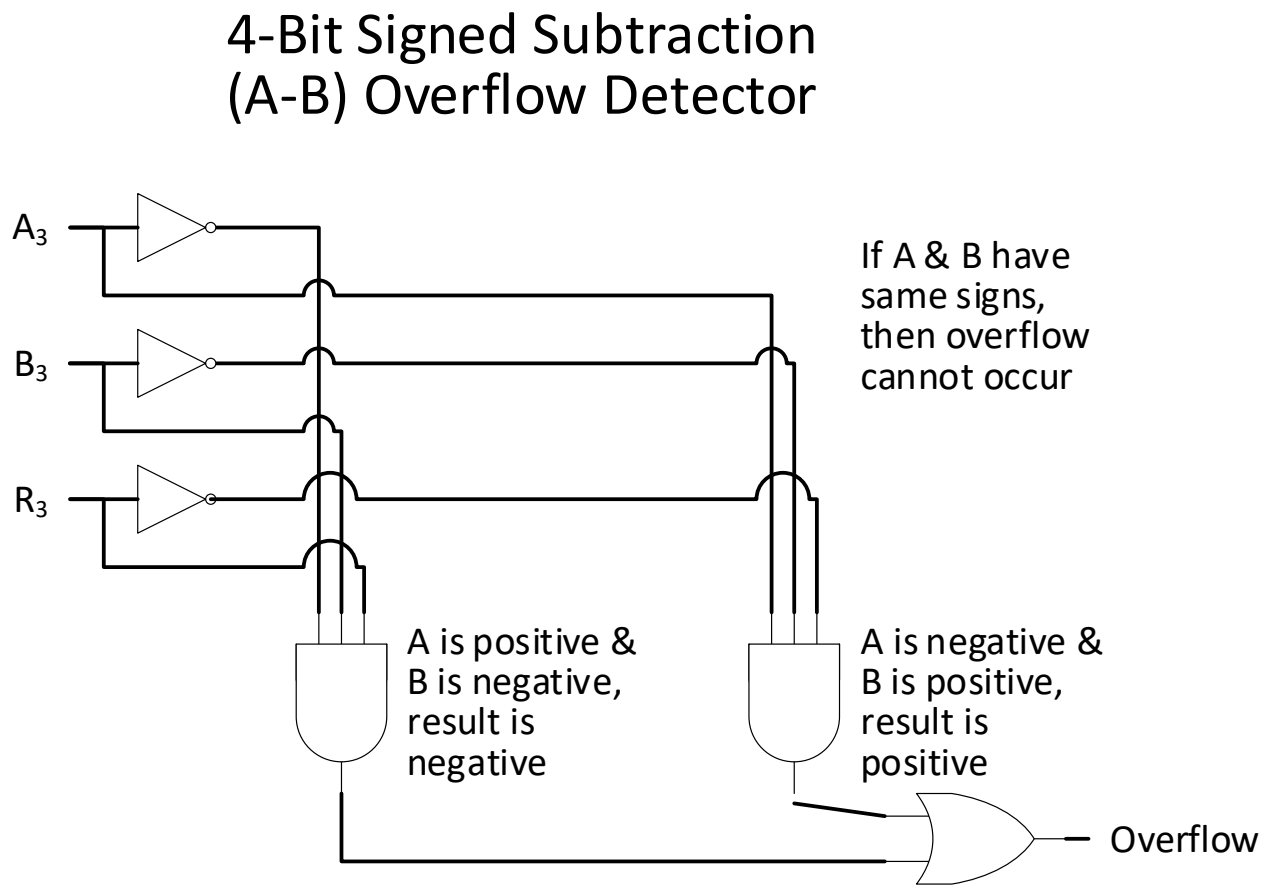
- minuend – subtrahend

- Signed overflow occurs from an adder while performing subtraction when
  - The operands are of opposite signs, but the result does not appear to have the same sign as the minuend

    Or, in other words:
  - The minuend is positive and the subtrahend is negative, but the result is negative; or
  - The minuend is negative and the subtrahend is positive, but the result is positive

- Signed subtraction cannot overflow when
  - The operands are of the same sign

Circuit to Detect Overflow from a 4-bit Subtractor



MIPS **sltu** and **slt** Instructions

- MIPS instructions
  - sltu rd, rs, rt
  - slt rd, rs, rt

- Behavior
  - SLT: Set on Less Than
    - Compare as signed 32-bit integers
    - Result is 1 if true, 0 if false
  - SLTU: Set on Less Than Unsigned
    - Compare as unsigned 32-bit integers
    - Result is 1 if true, 0 if false

- GPR[rd] <- (GPR[rs] < GPR[rt])

Unsigned comparison: **sltu**

- result is 1 if: GPR[rs] <UNSIGNED GPR[rt] and 0 otherwise
- GPR[rs] - GPR[rt] <UNSIGNED GPR[rt] - GPR[rt]
- GPR[rs] - GPR[rt] <UNSIGNED 0
- result is 1 if: GPR[rs] + ~GPR[rt] + 1 <UNSIGNED 0 and 0 otherwise

- Let's look at several values near each other in magnitude

Unsigned Comparison: 3<2 (0003 < 0002)

- In hexadecimal:
- 0003 < 0002 ?
- 0003 - 0002
- 0003 + ~0002 + 1
- 0003 + FFFD + 1
- result == 0001
- carryOutMSB == 1
- FALSE: 3 < 2

Unsigned Comparison: 3<3 (0003 < 0003)

- In hexadecimal:
- 0003 < 0003 ?
- 0003 - 0003
- 0003 + ~0003 + 1
- 0003 + FFFC + 1
- result == 0000
- carryOutMSB == 1
- FALSE: 3 < 3

Unsigned Comparison: 3<4 (0003 < 0004)

- In hexadecimal:
- 0003 < 0004 ?
- 0003 - 0004
- 0003 + ~0004 + 1
- 0003 + FFFB + 1
- result == FFFF
- carryOutMSB == 0
- TRUE: 3 < 4

Unsigned Comparison: 65535<0 (FFFF < 0000)

- In hexadecimal:
- FFFF < 0000 ?
- FFFF - 0000
- FFFF + ~0000 + 1
- FFFF + FFFF + 1
- result == FFFF
- carryOutMSB == 1
- FALSE: 65535 < 0

Unsigned Comparison: 65535<65535 (FFFF < FFFF)

- In hexadecimal:
- FFFF < FFFF ?
- FFFF - FFFF
- FFFF + ~FFFF + 1
- FFFF + 0000 + 1
- result == 0000
- carryOutMSB == 1
- FALSE: 65535 < 65535

Unsigned Comparison: 65535<65534 (FFFF < FFFE)

- In hexadecimal:
- FFFF < FFFE ?
- FFFF - FFFE
- FFFF + ~FFFE + 1
- FFFF + 0001 + 1
- result == 0001
- carryOutMSB == 1
- FALSE: 65535 < 65534

Unsigned Comparison: 0<65535 (0000 < FFFF)

- In hexadecimal:
- 0000 < FFFF ?
- 0000 - FFFF
- 0000 + ~FFFF + 1
- 0000 + 0000 + 1
- result == 0001
- carryOutMSB == 0
- TRUE: 0 < 65535

Unsigned Comparison: 0<0 (0000 < 0000)

- In hexadecimal:
- 0000 < 0000 ?
- 0000 - 0000
- 0000 + ~0000 + 1
- 0000 + FFFF + 1
- result == 0000
- carryOutMSB == 1
- FALSE: 0 < 0

Unsigned Comparison: 0<1 (0000 < 0001)

- In hexadecimal:
- 0000 < 0001 ?
- 0000 - 0001
- 0000 + ~0001 + 1
- 0000 + FFFE + 1
- result == FFFF
- carryOutMSB == 0
- TRUE: 0 < 1

Using Subtraction to Determine Result of **sltu**

- So, if you're trying to determine the answer for
  - sltu rd, rs, rt
- by computing
  - GPR[rs] - GPR[rt]
- by using an adder to compute
  - GPR[rs] + ~GPR[rt] + 1
- then, sltu's result is 1 when carryOutMSB == 0 and 0 otherwise

- Reminder: This is performing subtraction by adding the two's-complement of the value being subtracted

PDP-11 Compare Instructions

- PDP-11 Compare Instructions
  - CMP source, destination
    - 16-bit word version
  - CMPB source, destination
    - 8-bit byte version
- CMP(B) is executed as if src -SIGNED dest
  - The PDP-11 has a subtractor in addition to an adder
    - The C (Carry) flag and the V (oVerflow) flag are set appropriately for a subtractor (not an adder)
  - As a result,
    - C (Carry) flag is set if there is a ***borrow*** from the MSB
      - *i.e.,* if (src + ~dest + 1) < 2^16
    - V (oVerflow) flag is set if there is arithmetic overflow
      - *i.e.,* if operands are of opposite signs and the sign of the destination is the same as the sign of the result
    - Z (Zero) flag is set if the result == 0
    - N (Negative) flag is set if the result < 0

PDP-11 Unsigned Branch Instructions

- The following instructions would follow a CMP or CMPB instruction
- PDP-11 Unsigned Branch Instructions
  - BHI Branch if higher
    - Branch if source >UNSIGNED destination
    - Branch if the previous operation causes neither a carry nor a zero (equal) result: !c & !z
  - BLOS Branch if lower or same
    - Branch if source <=UNSIGNED destination
    - Branch if the previous operation causes carry to be set or zero to be set: c | z
  - BHIS Branch if higher or same
    - Branch if source >=UNSIGNED destination
    - Branch if the previous operation causes carry to be 0: !c
  - BLO Branch if lower
    - Branch if source <UNSIGNED destination
    - Branch if the previous operation causes carry to be set: c

Signed comparison: **slt**

- result is 1 if: GPR[rs] < GPR[rt] and 0 otherwise
- GPR[rs] - GPR[rt] < GPR[rt] - GPR[rt]
- GPR[rs] - GPR[rt] < 0
- result is 1 if: GPR[rs] + ~GPR[rt] + 1 < 0 and 0 otherwise

- We know that comparing for "< 0" is the same as checking the sign bit
- But, the problem is that overflows can occur and they can cause the sign bit to not indicate the sign of the result

- The appropriate Boolean expressions for the PDP-11 follow

PDP-11 Signed Branch Instructions

- The following instructions would follow a CMP or CMPB instruction
- PDP-11 Signed Branch Instructions
  - BGE Branch if greater-than or equal
    - Branch if source >=SIGNED destination
    - Branch if the previous operation causes either both N and V clear or both set:
      (n == v) or (n eqv v)
  - BGT Branch if greater-than
    - Branch if source >SIGNED destination
    - Branch if the previous operation causes Z to be clear and N is that same as V:
      (!z & (n == v)) or (!z & (n eqv v))
  - BLE Branch if less-than or equal
    - Branch if source <=SIGNED destination
    - Branch if the previous operation causes Z to be set or if N does not equal V:
      z | (n xor v)
  - BLT Branch if less-than
    - Branch if source <SIGNED destination
    - Branch if the previous operation causes N xor V to be 1:
      n xor v

Patterson & Hennessy: Computer Organization & Design, MIPS Edition, 6th Ed.

- Appendix B, Section B.5, page B-721: Tailoring the 32-Bit ALU to MIPS
  - Starts discussion about **`slt`** and **`sltu`** instructions
- Appendix B, Section B.5, Figure B.5.10, page B-723: Bit-slice ALU to perform AND, OR, addition, and set-on… instructions
  - Top schematic shows how to use the output mux in the bit-slice ALU to select an input named “Less” for set-on-… instructions
  - The result of a set-on- instruction should have the value 0 or 1
  - So, clearly the least-significant bit (LSB) of the ALU should be either 0 or 1 depending on the false or true result; All other bits should always be 0

Signed Comparison: 3<2 (0003 < 0002)

- In hexadecimal:
- 0003 < 0002 ?
- 0003 - 0002
- 0003 + ~0002 + 1
- 0003 + FFFD + 1
- result == 0001
- carryOutMSB == 1
- $result_{MSB}$ = 0
- FALSE: 3 < 2

Signed Comparison: 3<3 (0003 < 0003)

- In hexadecimal:
- 0003 < 0003 ?
- 0003 - 0003
- 0003 + ~0003 + 1
- 0003 + FFFC + 1
- result == 0000
- carryOutMSB == 1
- $result_{MSB}$ = 0
- FALSE: 3 < 3

Signed Comparison: 3<4 (0003 < 0004)

- In hexadecimal:
- 0003 < 0004 ?
- 0003 - 0004
- 0003 + ~0004 + 1
- 0003 + FFFB + 1
- result == FFFF
- carryOutMSB == 0
- $result_{MSB}$ = 1
- TRUE: 3 < 4

Signed Comparison: -1<0 (FFFF < 0000)

- In hexadecimal:
- FFFF < 0000 ?
- FFFF - 0000
- FFFF + ~0000 + 1
- FFFF + FFFF + 1
- result == FFFF
- carryOutMSB == 1
- $result_{MSB}$ = 1
- TRUE: -1 < 0

Signed Comparison: -1<-1 (FFFF < FFFF)

- In hexadecimal:
- FFFF < FFFF ?
- FFFF - FFFF
- FFFF + ~FFFF + 1
- FFFF + 0000 + 1
- result == 0000
- carryOutMSB == 1
- $result_{MSB}$ = 0
- FALSE: -1 < -1

Signed Comparison: -1<-2 (FFFF < FFFE)

- In hexadecimal:
- FFFF < FFFE ?
- FFFF - FFFE
- FFFF + ~FFFE + 1
- FFFF + 0001 + 1
- result == 0001
- carryOutMSB == 1
- $result_{MSB}$ = 0
- FALSE: -1 < -2

Signed Comparison: 0<-1 (0000 < FFFF)

- In hexadecimal:
- 0000 < FFFF ?
- 0000 - FFFF
- 0000 + ~FFFF + 1
- 0000 + 0000 + 1
- result == 0001
- carryOutMSB == 0
- $result_{MSB}$ = 0
- FALSE: 0 < -1

Signed Comparison: 0<0 (0000 < 0000)

- In hexadecimal:
- 0000 < 0000 ?
- 0000 - 0000
- 0000 + ~0000 + 1
- 0000 + FFFF + 1
- result == 0000
- carryOutMSB == 1
- $result_{MSB}$ = 0
- FALSE: 0 < 0

Signed Comparison: 0<1 (0000 < 0001)

- In hexadecimal:
- 0000 < 0001 ?
- 0000 - 0001
- 0000 + ~0001 + 1
- 0000 + FFFE + 1
- result == FFFF
- carryOutMSB == 0
- $result_{MSB} = 1$
- TRUE: 0 < 1

Using Subtraction to Determine Result of **slt**

- So, if you're trying to determine the answer for
  - slt rd, rs, rt
- by computing
  - GPR[rs] - GPR[rt]
- by using an adder to compute
  - GPR[rs] + ~GPR[rt] + 1
- then, slt's result is 1 when $result_{MSB}$ and 0 otherwise

- Reminder: This is performing subtraction by adding the two's-complement of the value being subtracted