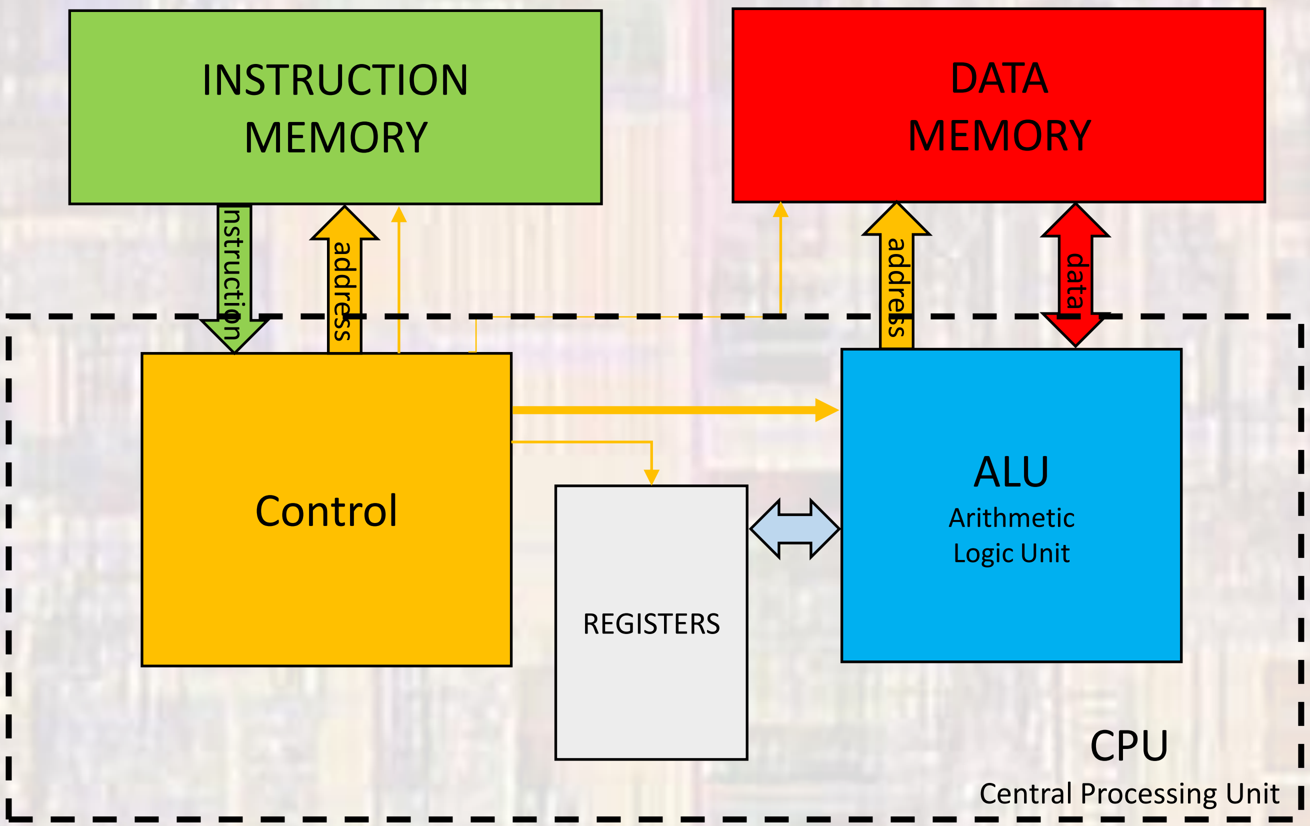
Single Cycle Processor ALU

Last updated 7/18/23

Single Cycle Processor - ALU

- Generalized Structure



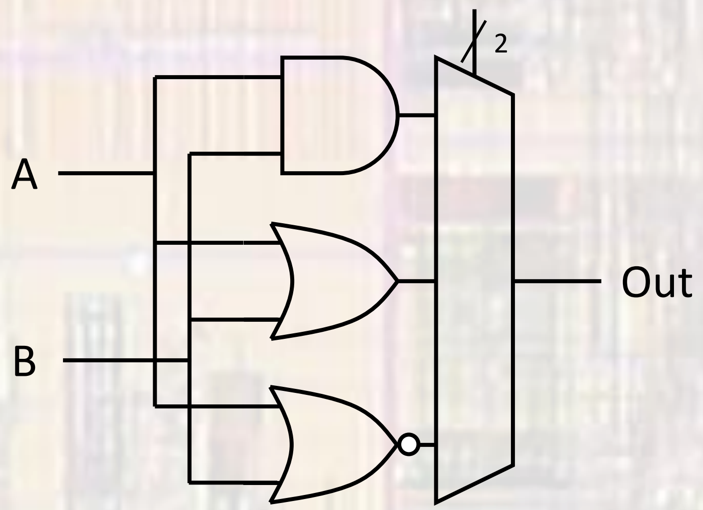
Single Cycle Processor - ALU

- Instruction Set
  - Instruction Set Architecture - ISA
  - Arithmetic Instructions
    - Add
    - Subtract
    - Less Than
  - Logical Instructions
    - AND
    - OR
    - NOR

Single Cycle Processor - ALU

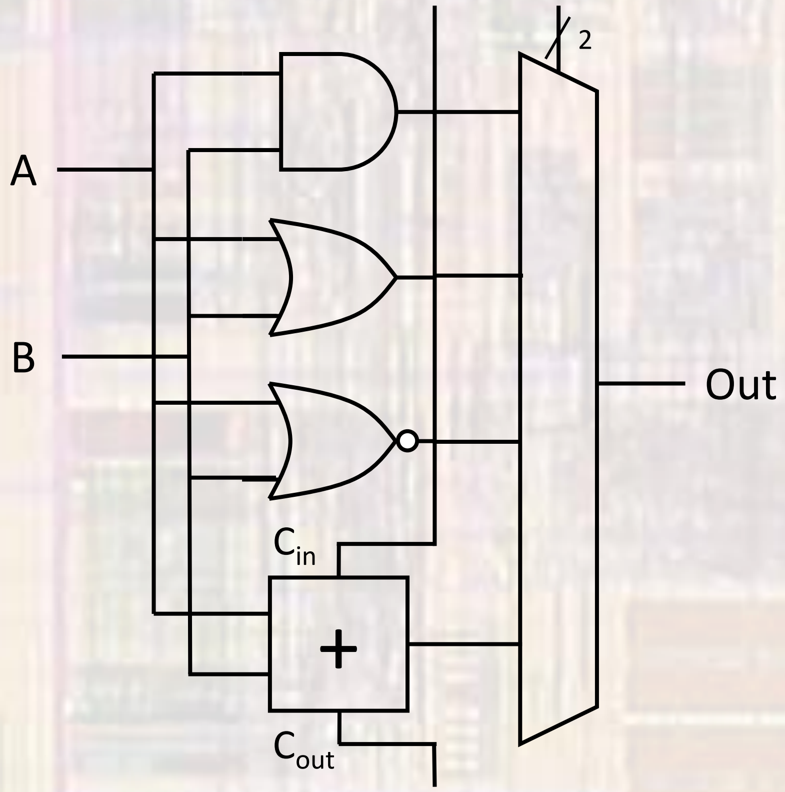
- ALU - Implementation

  - Logical Instructions
    - AND, OR, NOR
  - 2 inputs A and B
  - 1 output



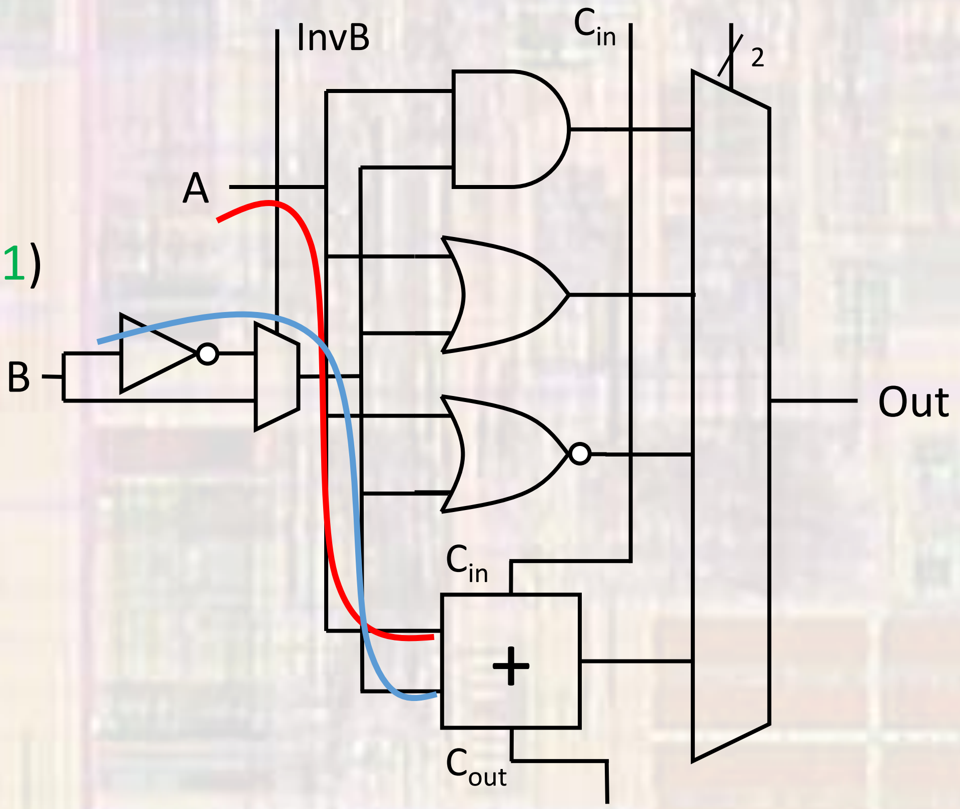
Single Cycle Processor - ALU

- ALU – Implementation
  - Arithmetic Instructions
    - ADD
    - Inputs: A, B, C_{in}
    - Outputs: Out, C_{out}



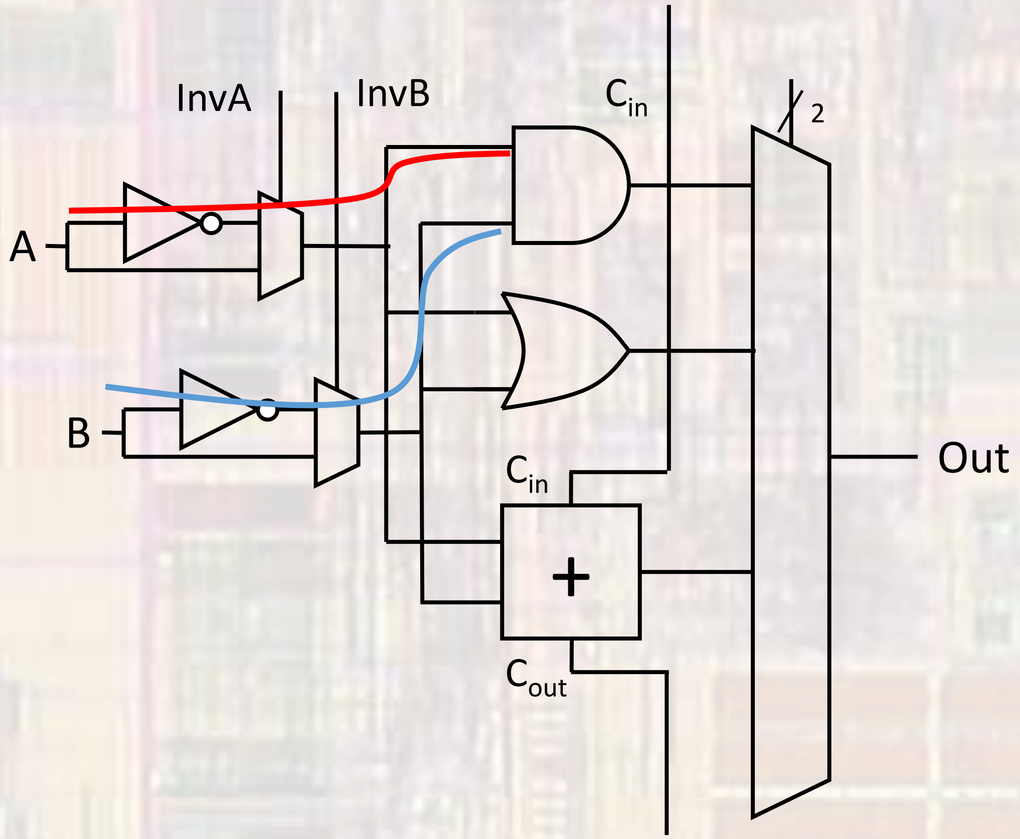
Single Cycle Processor - ALU

- ALU – Implementation
  - Arithmetic Instructions
    - SUB (2's complement)
    - $A - B = A + (-B)$
    - $= A + (\overline{B} + 1)$
    - Invert B and add 1 ($C_{inB0} = 1$)
    - Inputs: A, B, C_{in}
    - Outputs: Out, C_{out}



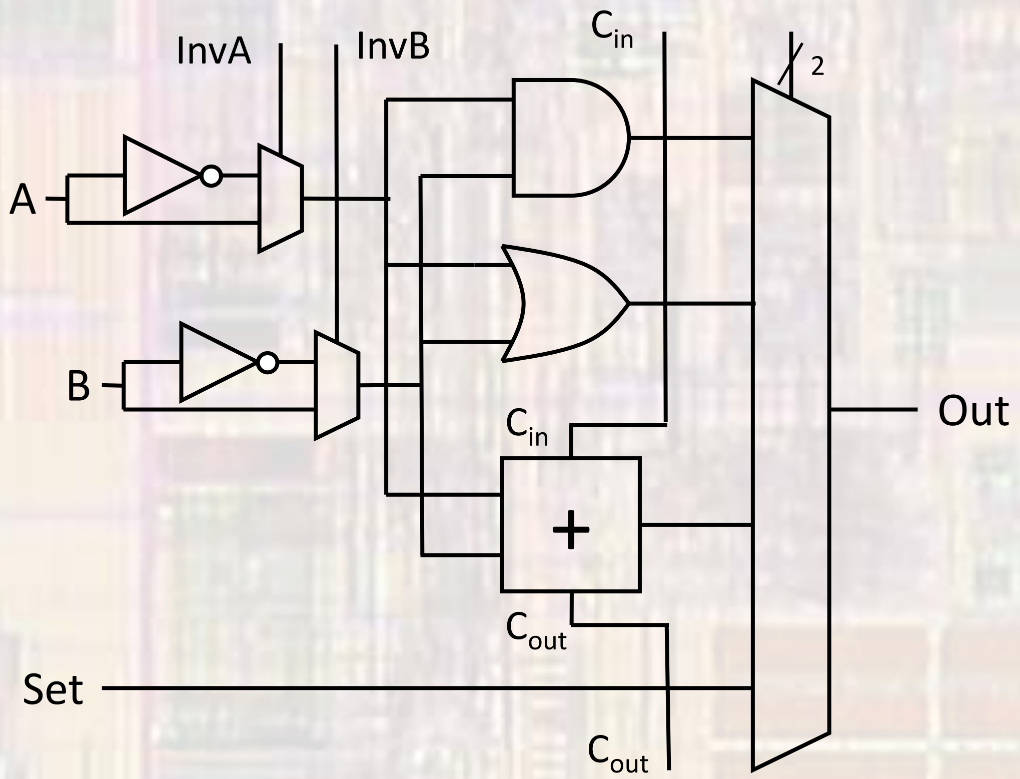
Single Cycle Processor - ALU

- ALU - Implementation
  - Revisit NOR
    - $\overline{A + B} = \bar{A}\,\bar{B}$



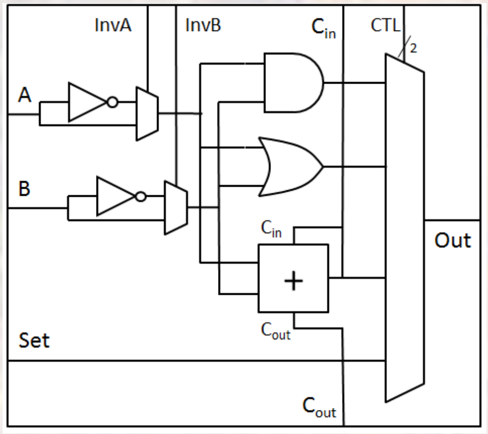
Single Cycle Processor - ALU

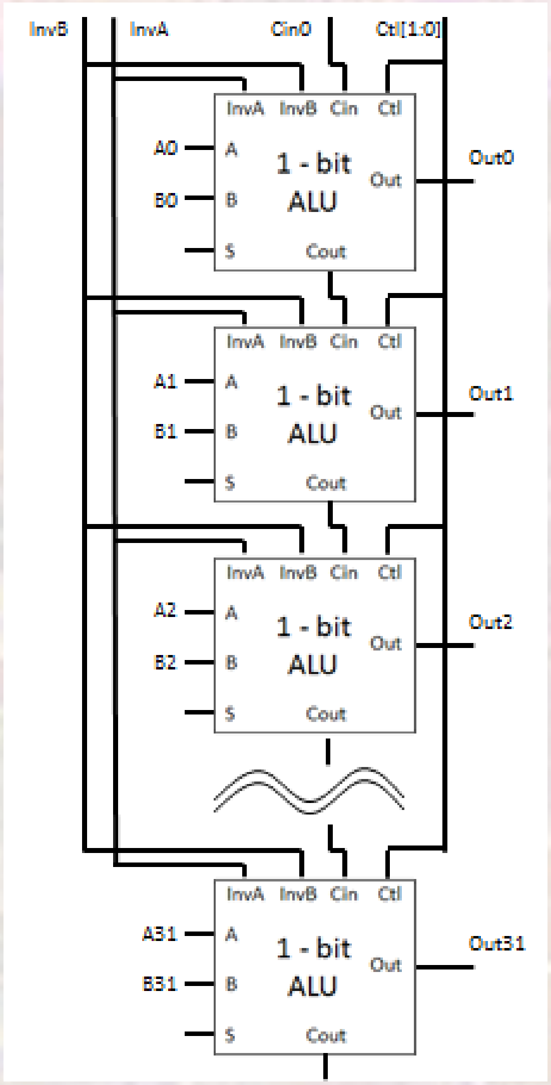
- ALU - Implementation
  - Pre-plan for set function



Single Cycle Processor - ALU

- ALU - Implementation
  - 32 bits





Single Cycle Processor - ALU

- ALU – Implementation

  - Determine if A < B
    - A < B → (A – B) < 0 → negative answer
      - Subtraction is implemented by addition
      - A – B → A + $\overline{B}$ + 1

    - MSB indicates sign in 2's complement arithmetic
      - MSB = 1 → negative number
      - MSB = 0 → positive number

  - Set On Less Than instruction
    - Use Adder MSB for SLT signal
    - If A < B: Out[31:1] = 0, Out[0] = 1
    - If A ≥ B: Out[31:0] = 0, Out[0] = 0
    - SET = 1

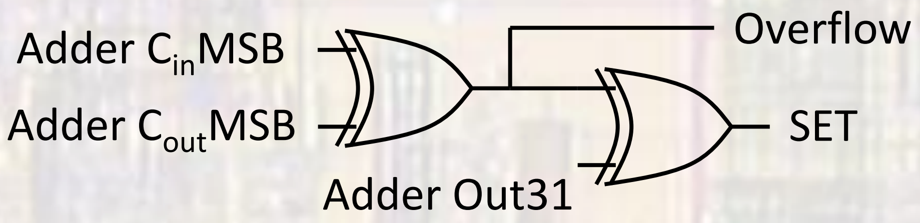
Single Cycle Processor - ALU

- ALU – Implementation
  - Set On Less Than – cont'd
    - MSB after subtraction indicates sign
      - MSB = 1 → negative number
      - MSB = 0 → positive number
    - Exception: Subtraction (addition) is not valid if overflow occurs

      If overflow occurs, MSB is wrong sign

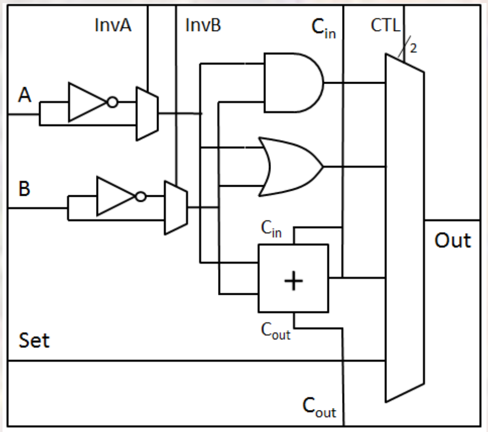
      SET becomes MSB xor OVERFLOW

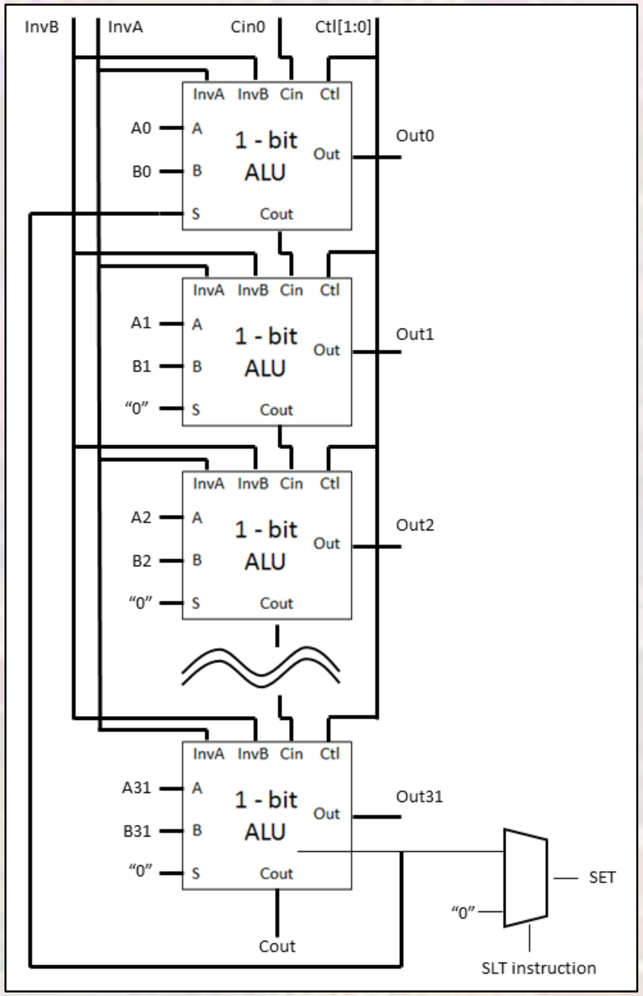
Not implementing this function



Single Cycle Processor - ALU

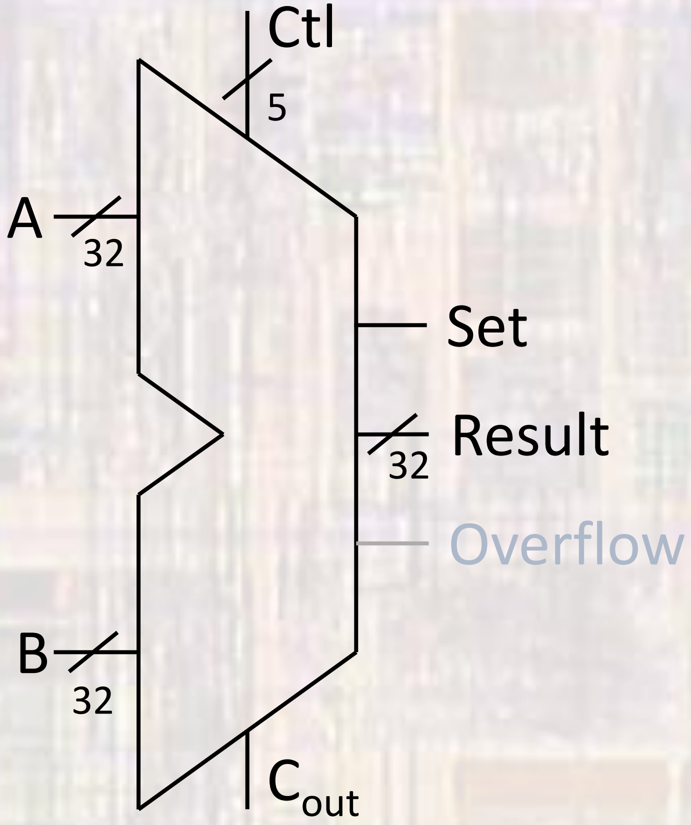
- ALU - Implementation

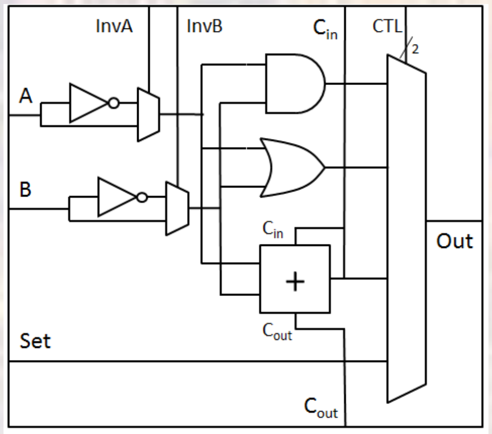




Single Cycle Processor - ALU

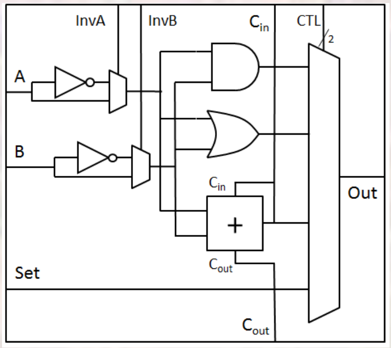
- ALU – Implementation
  - Control
    - invA
    - invB
    - Cin
    - ctl[1:0]





Single Cycle Processor - ALU

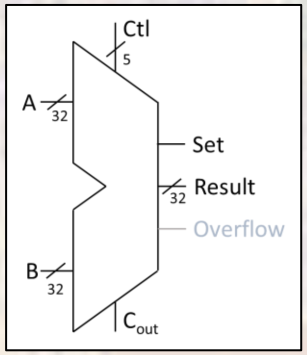
- ALU – Implementation



DeMorgan

Addition

Operation	invA	invB	Cin	ctl[1]	ctl[0]
AND	0	0	x	1	1
OR	0	0	x	1	0
NOR	1	1	x	1	1
ADD	0	0	0	0	1
SUB	0	1	1	0	1
SLT	0	1	1	0	0



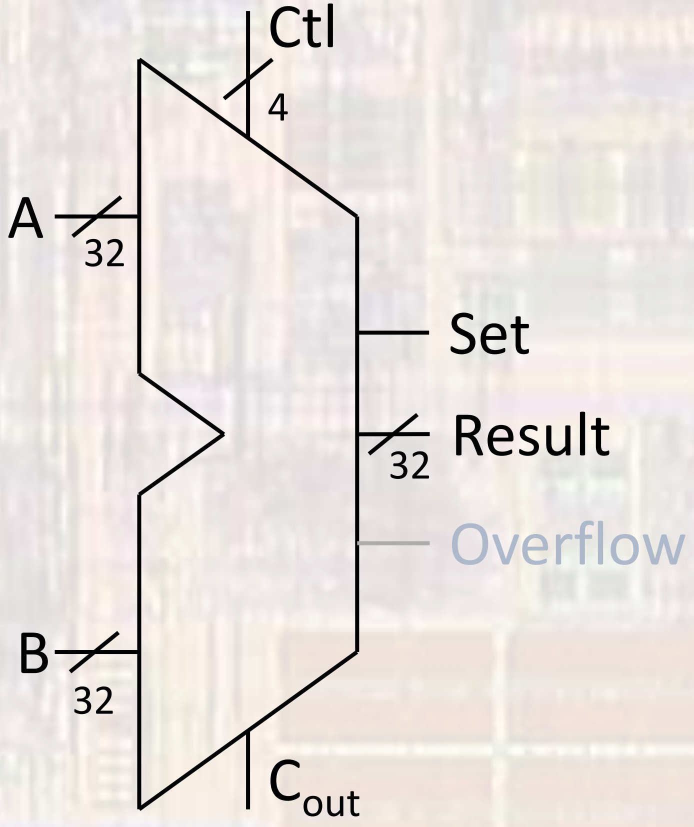
What other logic function do we get for free

Single Cycle Processor - ALU

- ALU – Implementation
  - Note: C_{in} and invB can always be the same → combine (negB)

    Reduces control lines to 4

Operation	invA	negB	ctl[1]	ctl[0]
AND	0	0	1	1
OR	0	0	1	0
NOR	1	1	1	1
ADD	0	0	0	1
SUB	0	1	0	1
SLT	0	1	0	0



Single Cycle Processor - ALU

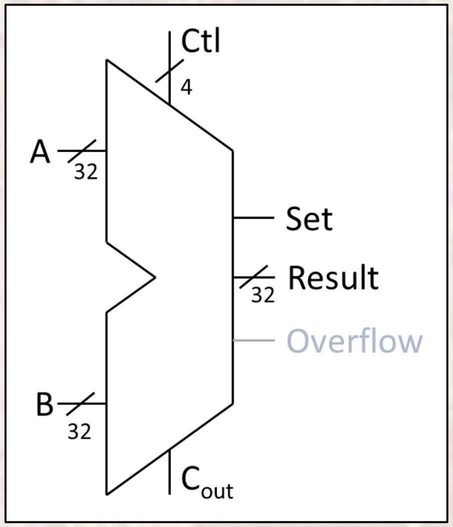
- ALU – Implementation

  - 3 out of 6 instructions involve addition

Current implementation is very slow – why?

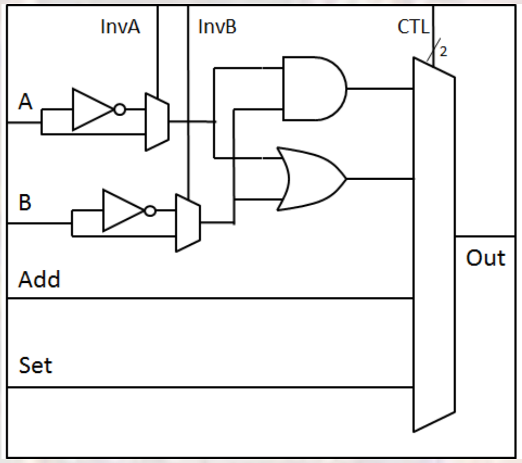
Operation	invA	negB	ctl[1]	ctl[0]
AND	0	0	1	1
OR	0	0	1	0
NOR	1	1	1	1
ADD	0	0	0	1
SUB	0	1	0	1
SLT	0	1	0	0

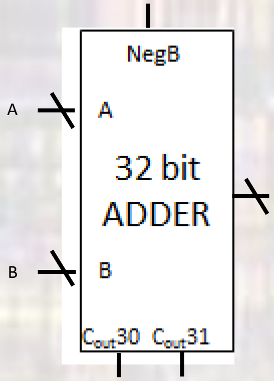
Addition (ADD, SUB, SLT)

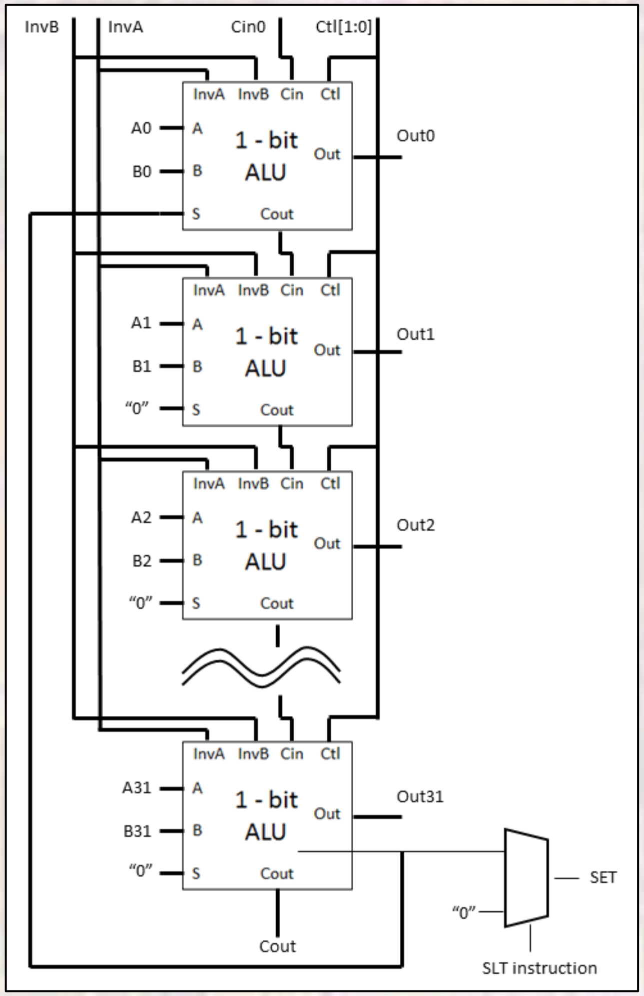


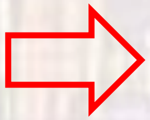
Single Cycle Processor - ALU

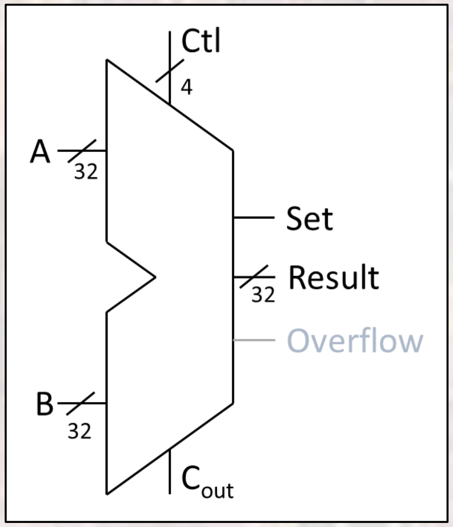
- Enhanced Adder


InvA
InvB
CTL
2
A
B
Add
Set
Out


NegB
A
A
32 bit
ADDER
B
B
C_{out}30
C_{out}31


InvB
InvA
Cin0
Ctl[1:0]
InvA InvB Cin Ctl
A0
B0
A
B
S
1 - bit
ALU
Out
Cout
Out0
A1
B1
"0"
Out1
A2
B2
"0"
Out2
A31
B31
"0"
Out31
Cout
"0"
SET
SLT instruction




Ctl
4
A
32
B
32
Set
Result
32
Overflow
C_{out}