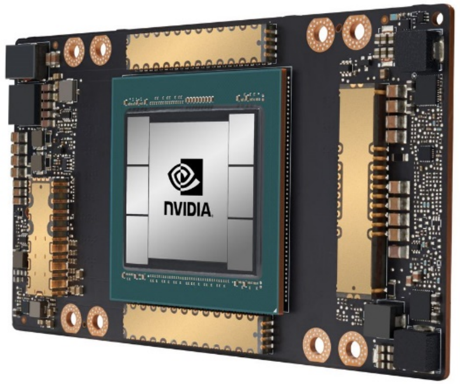
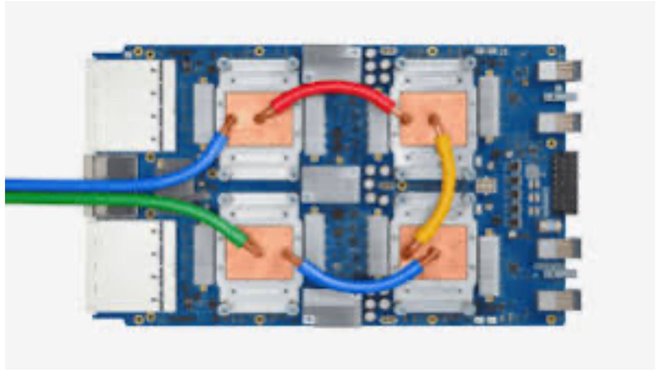
CSCB58:
Computer Organization



Prof. Gennady Pekhimenko

University of Toronto

Fall 2020



The content of this lecture is adapted from the lectures of Larry Zheng and Steve Engels

CSCB58 Week 7: Summary

Week 7 Summary

We learned

- Circuit Efficiency
  - Propagation and contamination delays
- Processor components
  - ALUs

Question #1

- What is the result of the following operation?

```
   1010
 x 1101
 ------
```

```
       1010
 x     1101
 ----------
       0000
      1101
     0000
    1101
 ----------
   10000010
```

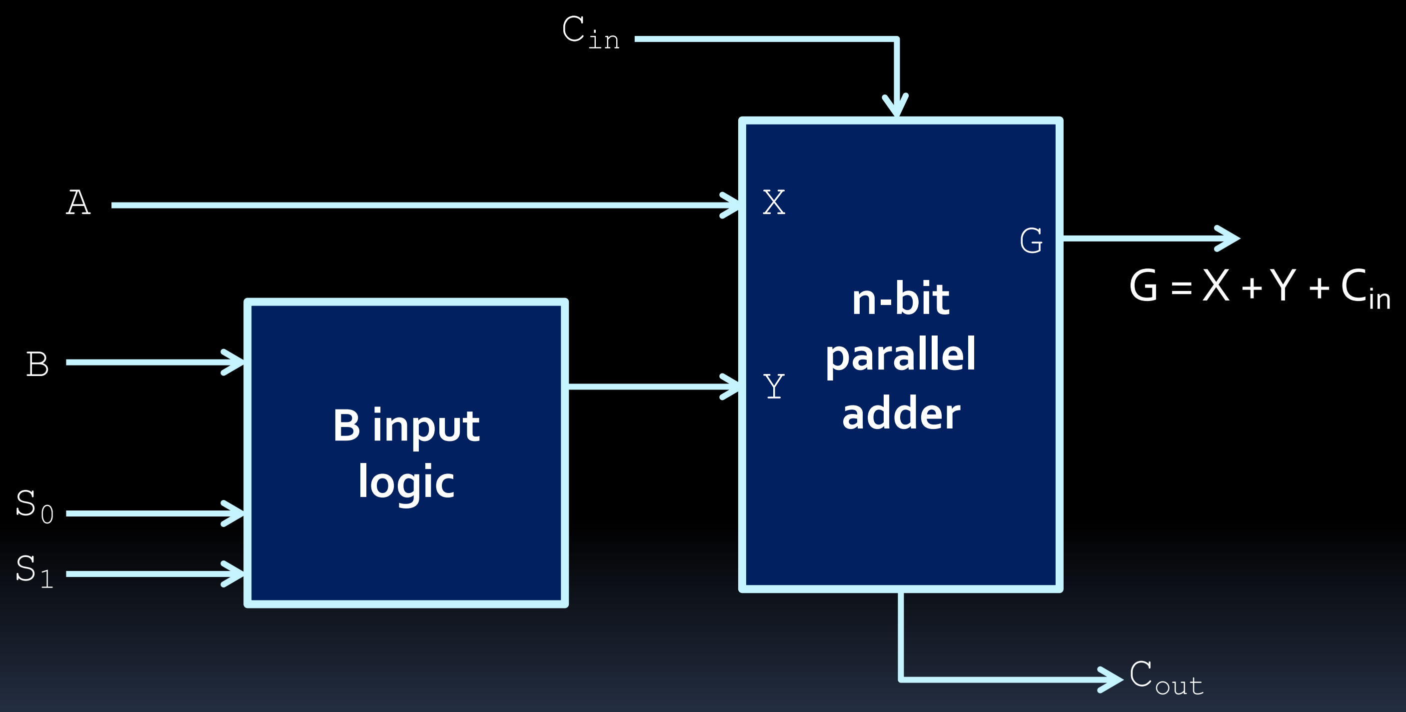
Verify!

→ 10
→ 13

→ 130

Question #2

- The arithmetic unit of the ALU looks like this:



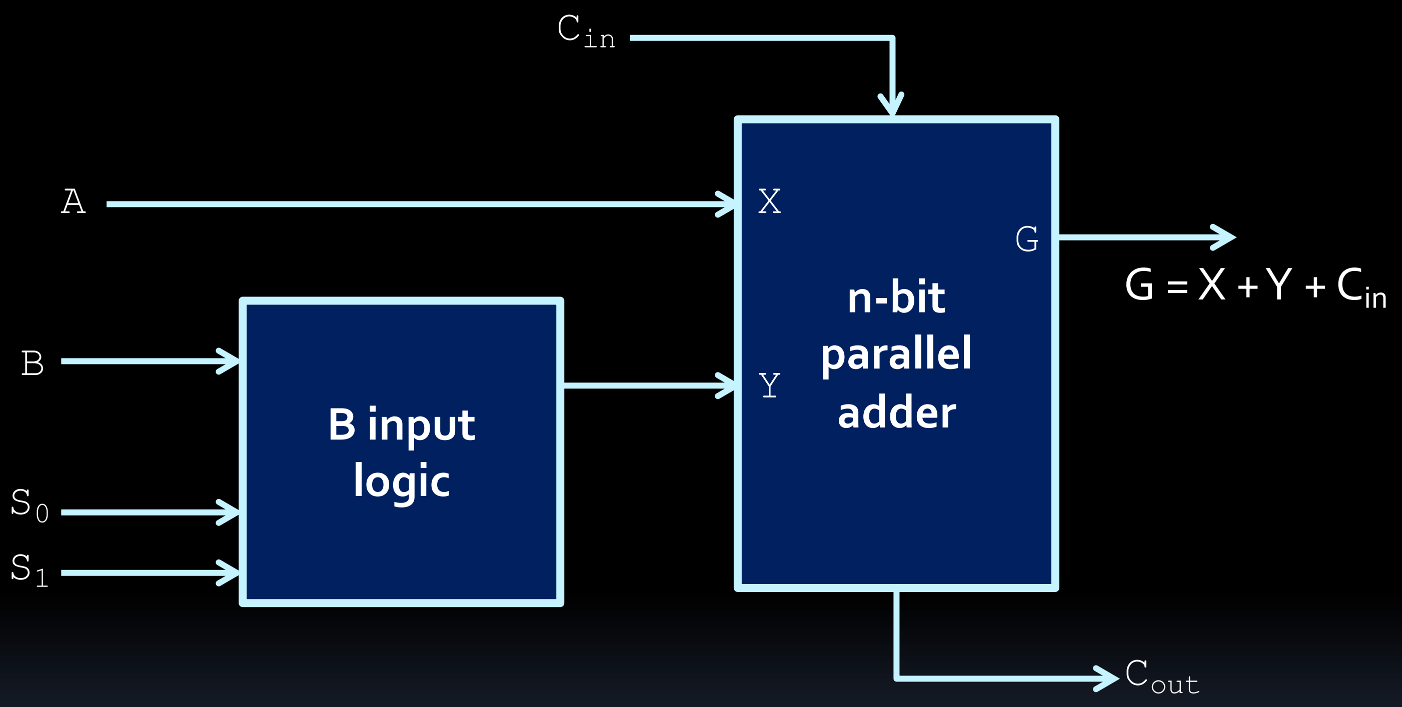
- What values for S_0, S_1 and C_{in} do we need in order to subtract B from A?

Question #2 (cont'd)

- Kind of an unfair question, in that there's a table that fills in some necessary details:

Select		Input	Operation	
S_1	S_0	Y	$C_{in}=0$	$C_{in}=1$
0	0	All 0s	G = A (transfer)	G = A+1 (increment)
0	1	B	G = A+B (add)	G = A+B+1
1	0	$\overline{B}$	G = A+$\overline{B}$	G = A+$\overline{B}$+1 (subtract)
1	1	All 1s	G = A-1 (decrement)	G = A (transfer)

Question #2 (cont'd)



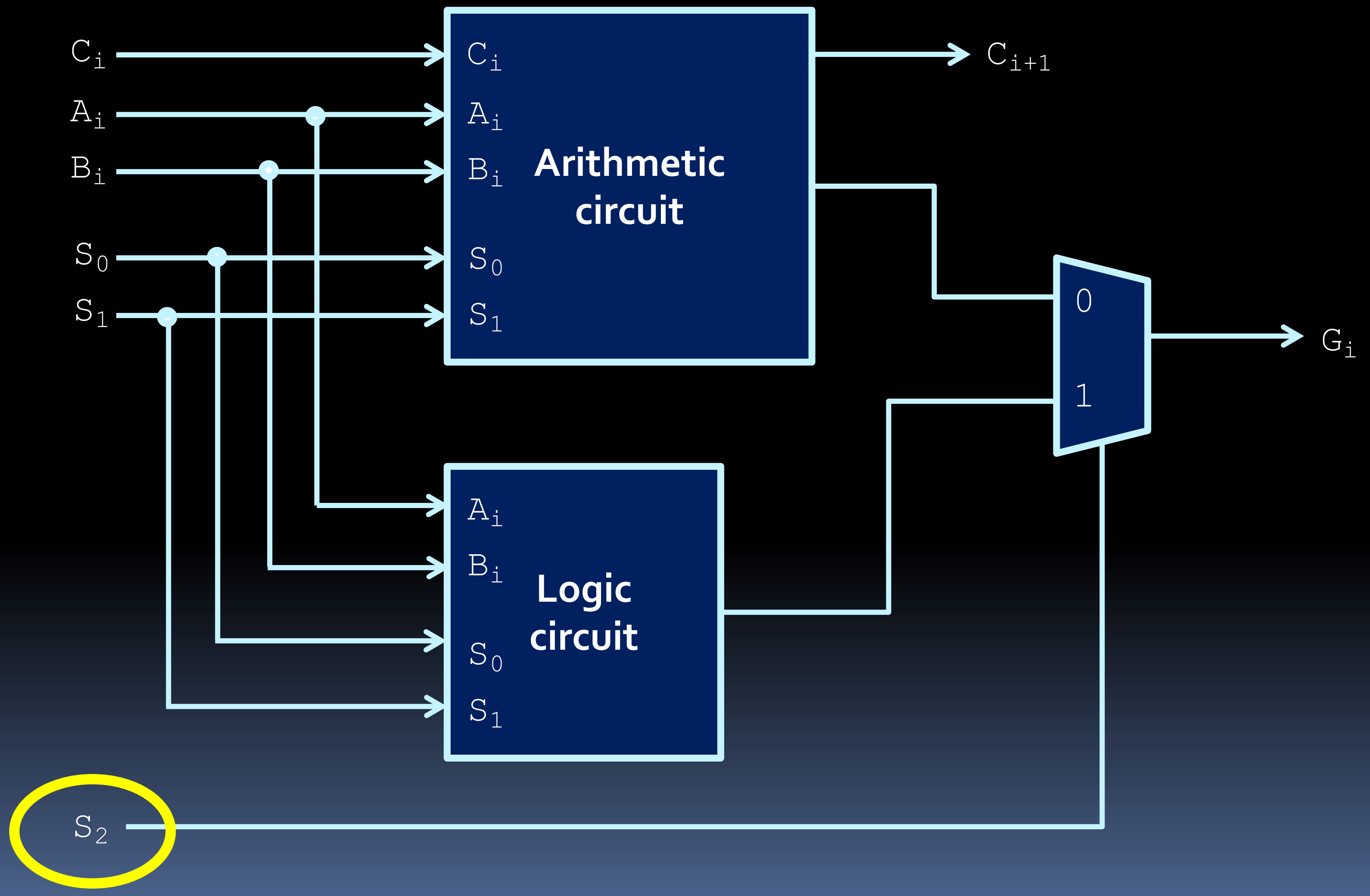
- To subtract B from A, you must set S_0=0, S_1=1 and C_{in}=1.

Question #3

- In an ALU, S_0 and S_1 determine which kind of arithmetic or logical function to perform. But there are 3 select signals that go into the ALU.

  What does S_2 do?

Question #3 (cont'd)



Booth's Algorithm

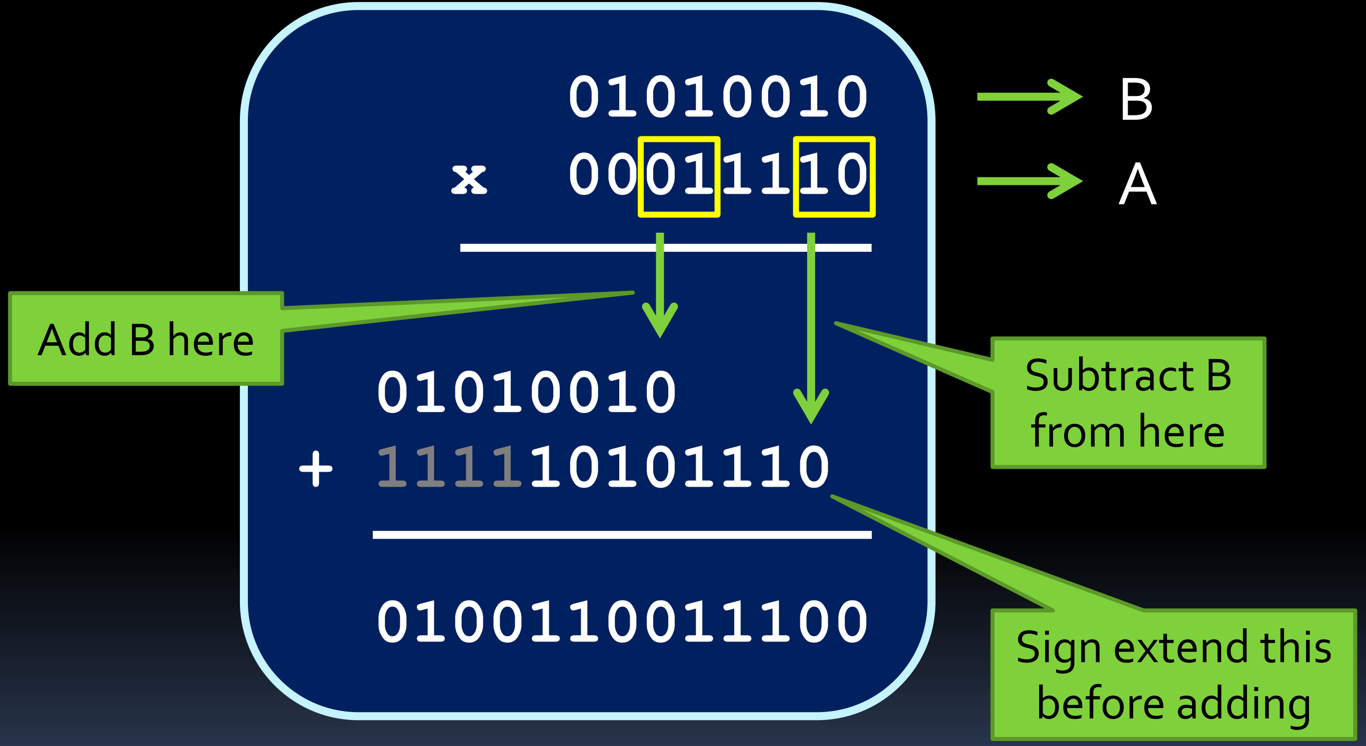
- Devised as a way to take advantage of circuits where shifting is cheaper than adding, or where space is at a premium.
  - Based on the premise that when multiplying by certain values (e.g. `99`), it can be easier to think of this operation as a difference between two products.
- Consider the shortcut method when multiplying a given decimal value `X` by `9999`:
  - `X*9999 = X*10000 - X*1`
- Now consider the equivalent problem in binary:
  - `X*001111 = X*010000 - X*1`

Booth's Algorithm

- This idea is triggered on cases where two neighboring digits in an operand are different.
  - If digits at `i` and `i-1` are `0` and `1`, the multiplicand is added to the result at position `i`.
  - If digits at `i` and `i-1` are `1` and `0`, the multiplicand is subtracted from the result at position `i`.
- The result is always a value whose size is the sum of the sizes of the two multiplicands.

Booth's Algorithm

- Example:



Booth’s Algorithm

- We need to make this work in hardware.
  - Option #1: Have hardware set up to compare neighbouring bits at every position in A, with adders in place for when the bits don’t match.
    - Problem: This is a lot of hardware, which Booth’s Algorithm is trying to avoid.
  - Option #2: Have hardware set up to compare two neighbouring bits, and have them move down through A, looking for mismatched pairs.
    - Problem: Hardware doesn’t move like that. Oops.

Booth's Algorithm

- Still need to make this work in hardware…
  - Option #3: Have hardware set up to compare two neighbouring bits in the lowest position of `A`, and looking for mismatched pairs in `A` by shifting `A` to the right one bit at a time.
    - Solution! This could work, but the accumulated solution `P` would have to shift one bit at a time as well, so that when `B` is added or subtracted, it's from the correct position.

Booth's Algorithm

Note: unlike the accumulator, the bits here are being shifted to the right!

- Steps in Booth's Algorithm:
  1. Designate the two multiplicands as A & B, and the result as some product P.
  2. Add an extra zero bit to the right-most side of A.
  3. Repeat the following for each original bit in A:
     a) If the last two bits of A are the same, do nothing.
     b) If the last two bits of A are `01`, then add B to the highest bits of P.
     c) If the last two bits of A are `10`, then subtract B from the highest bits of P.
     d) Perform one-digit arithmetic right-shift on both P and A.
  4. The result in P is the product of A and B.

Booth’s Algorithm Example

- Example: (-5) * 2

- Steps #1 & #2:
  - A = -5 → 11011
    - Add extra zero to the right → A = 11011 0
  - B = 2 → 00010
  - -B = -2 → 11110
  - P = 0 → 00000 00000

Booth's Algorithm Example

- Step #3 (repeat 5 times):
  - Check last two digits of A:

    1101 **10**

  - Since digits are 10, subtract B from the most significant digits of P:

    ```
     P     00000 00000
    -B    +11110
     P'    11110 00000
    ```

  - Arithmetic shift P and A one bit to the right:
    - A = 111011   P = 11111 00000

Booth’s Algorithm Example

- Step #3 (repeat 4 more times):
  - Check last two digits of A:

    1110 11

  - Since digits are 11, do nothing to P.
  - Arithmetic shift P and A one bit to the right:
    - A = 111101 P = 11111 10000

Booth’s Algorithm Example

- Step #3 (repeat 3 more times):
  - Check last two digits of A:

    ```
    1111 01
    ```

  - Since digits are 01, add B to the most significant digits of P:

    ```
     P     11111 10000
    +B    +00010
     P’    00001 10000
    ```

  - Arithmetic shift P and A one bit to the right:
    - A = 111110   P = 00000 11000

Booth's Algorithm Example

- Step #3 (repeat 2 more times):
  - Check last two digits of A:

```
1111 10
```

  - Since digits are 10, subtract B from the most significant digits of P:

```
 P     00000 11000
-B    +11110
 P'    11110 11000
```

  - Arithmetic shift P and A one bit to the right:
    - A = 111111 P = 11111 01100

Booth's Algorithm Example

- Step #3 (final time):
  - Check last two digits of A:

    1111 11

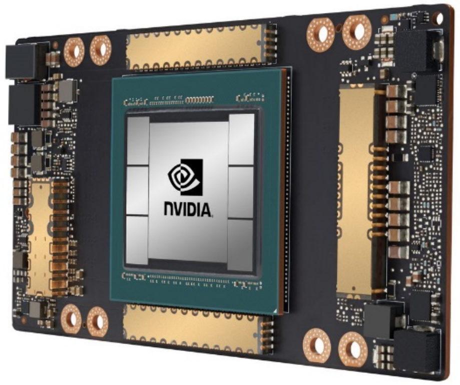
  - Since digits are 11, do nothing to P:
  - Arithmetic shift P and A one bit to the right:
    - A = 111111   P = 11111 10110

- Final product:

  P = 111110110

  = -10

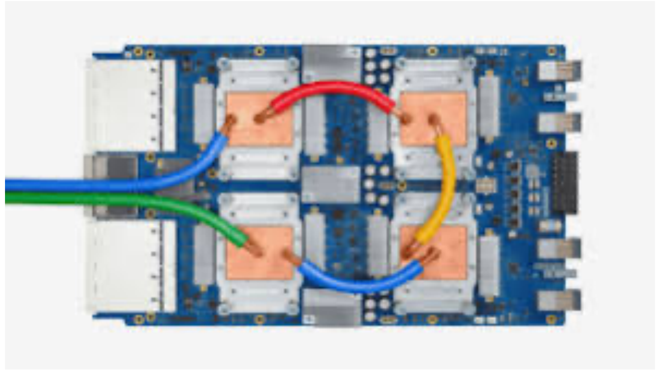
CSCB58: Computer Organization



Prof. Gennady Pekhimenko

University of Toronto

Fall 2020



The content of this lecture is adapted from the lectures of Larry Zheng and Steve Engels