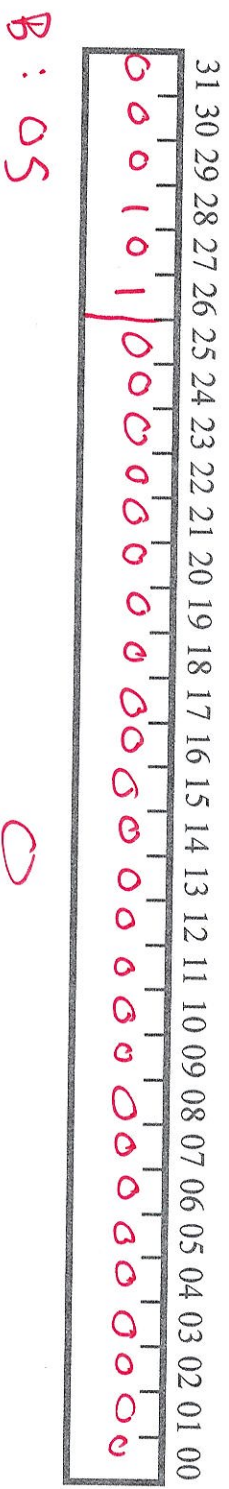


Review Problem 7

- ❖ Sometimes it can be useful to have a program loop infinitely. We can do that, regardless of location, by the instruction:
- ❖ LOOP: B ~~LOOP~~ ↻
- ❖ Convert this instruction to machine code



Conversion example

Compute the sum of the values $0 \dots N-1$

ADD X1, X31, X31

100:0101:1000	11111	000000	11111	000001
458	X31	0	X31	X1

ADD X2, X31, X31

100:0	101:1000	11111	000000	11111	00010
:	458	x31	0	x31	x2

~~TEST~~ + 3

[illegible]

TOP:

```
ADD X1, X1, X2
```

160:0101:1000	60010	6000000	00001	00061
:458	x2	0	x1	x1

```
ADDI X2, X2, #1
```

10:0100:0100	6000006000001	60010	60010
244	#1	X2	X2

TEST:

SUBS X31, X2, X0

11101011000	66666	6660000	666010	11111
758	$\times 0$	0	$\times 2$	$\times 31$

B.IT ~~TOP~~ -3

[illegible]

END:

Assembly & Machine Language

Assembly

Simple instructions

Mnemonics for humans

(Almost) 1-to-1 relationship w/machine language

Machine Language

Numeric representations of instructions

Fixed formats

Directly controls CPU hardware

Computer Arithmetic

Readings: 3.1-3.3, A.5

Review binary numbers, 2's complement

Develop Arithmetic Logic Units (ALUs) to perform CPU functions.

Introduce shifters, multipliers, etc.

Binary Numbers

Decimal: $469 = 4 \cdot 10^2 + 6 \cdot 10^1 + 9 \cdot 10^0$

Binary: $01101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (13)_{10}$

Example: $0111010101 = (?)_{10}$

$$\begin{array}{r} 1 + 4 + 16 + 64 + 128 + 256 \\ \hline 469 \end{array}$$

2's Complement Numbers

Positive numbers & zero have leading 0, negative have leading 1

Negation: Flip all bits and add 1
 13_{10}

$$\text{Ex: } -(01101)_2 = 10010 + 1 = 10011_2$$

To interpret numbers, convert to positive version, then convert:

$$\begin{aligned} 11010 &= -(11010) \\ &= -(00101 + 1) \\ &= -(00110) \\ &= -(6) \\ &= -6_{10} \end{aligned} \qquad \begin{aligned} 01100 &= \\ + 0100 &= \\ + 12_{10} & \end{aligned}$$

Sign Extension

Conversion of n-bit to (n+m)-bit 2's complement: replicate the sign bit

$$b_3b_2b_1b_0 = b_3b_3b_3b_2b_1b_0 = b_3b_3b_3b_3b_3b_3b_3b_3b_2b_1b_0$$

Ex - Convert to 8-bit: $01101 = (13)_{10}$

$$11101 = (-3)_{10}$$

00001101

convert?

11111101

$$= -(11111101)$$

$$= -(00000010 + 1)$$

$$= -(00000011)$$

$$= -3$$

Arithmetic Operations

Decimal:

$$\begin{array}{r} \text{Odometer} \rightarrow \text{with} \\ 57892 \\ + 78956 \\ \hline 36848 \end{array}$$

Binary:

$$\begin{array}{r} 000111 \\ + 0100101 \\ \hline 111100 \end{array}$$

Binary:

$$\begin{array}{r} 10100110 \\ - 00010111 \\ \hline \end{array}$$

flip the bits

$$\begin{array}{r} 11000001 \\ + 11161000 \\ \hline 10001111 \end{array}$$

Add one

Overflows

Operations can create a number too large for the number of bits

n-bit 2's complement can hold $-2^{(n-1)} \dots 2^{(n-1)-1}$

Can detect overflow in addition when highest bit has carry-in \neq carry-out

(carry-in) \oplus (carry-out) = 1

$$\begin{array}{r} 5 \\ 3 \\ \hline -8 \end{array}$$

$$\begin{array}{r} 0111 \\ 0101 \\ \hline 0011 \\ 1000 \end{array}$$

Overflow

$$\begin{array}{r} -7 \\ -2 \\ \hline 7 \end{array}$$

$$\begin{array}{r} 1000 \\ 1001 \\ \hline 1110 \\ 0111 \end{array}$$

Overflow

$$\begin{array}{r} 5 \\ 2 \\ \hline 7 \end{array}$$

$$\begin{array}{r} 0100 \\ 0101 \\ \hline 0010 \\ 0111 \end{array}$$

No overflow

$$\begin{array}{r} -3 \\ -5 \\ \hline -8 \end{array}$$

$$\begin{array}{r} 1111 \\ 1101 \\ \hline 1011 \\ 1000 \end{array}$$

No overflow

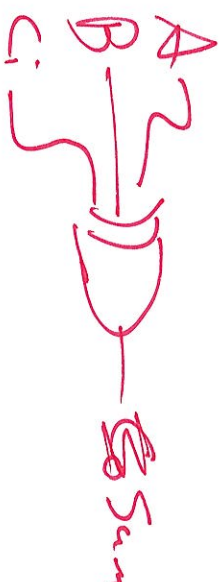
On an overflow, the top bit is flipped

Full Adder

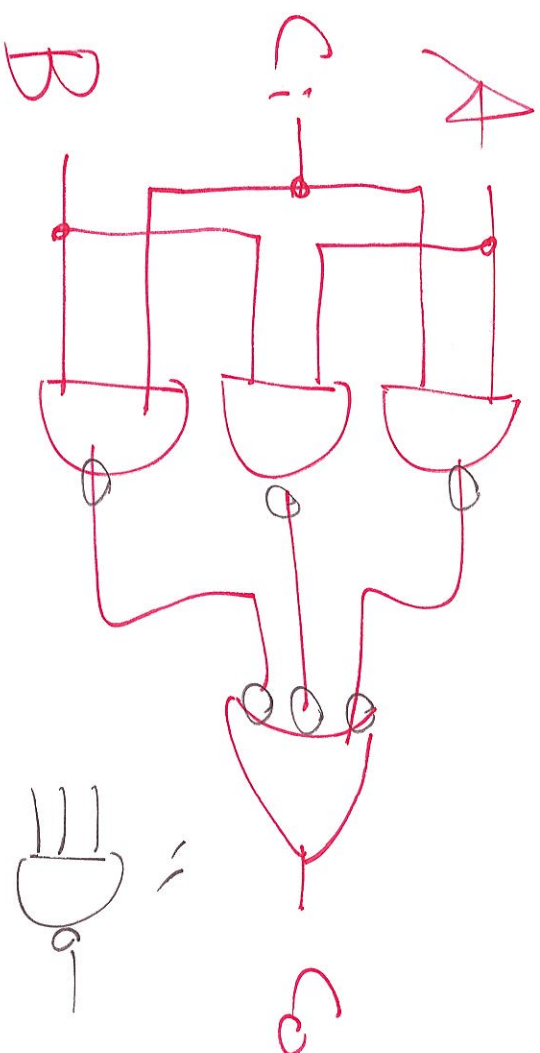
A	B	C _i	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A	B	C _i	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

AB (blue label)
 BC_i (blue label)
 AC_i (blue label)

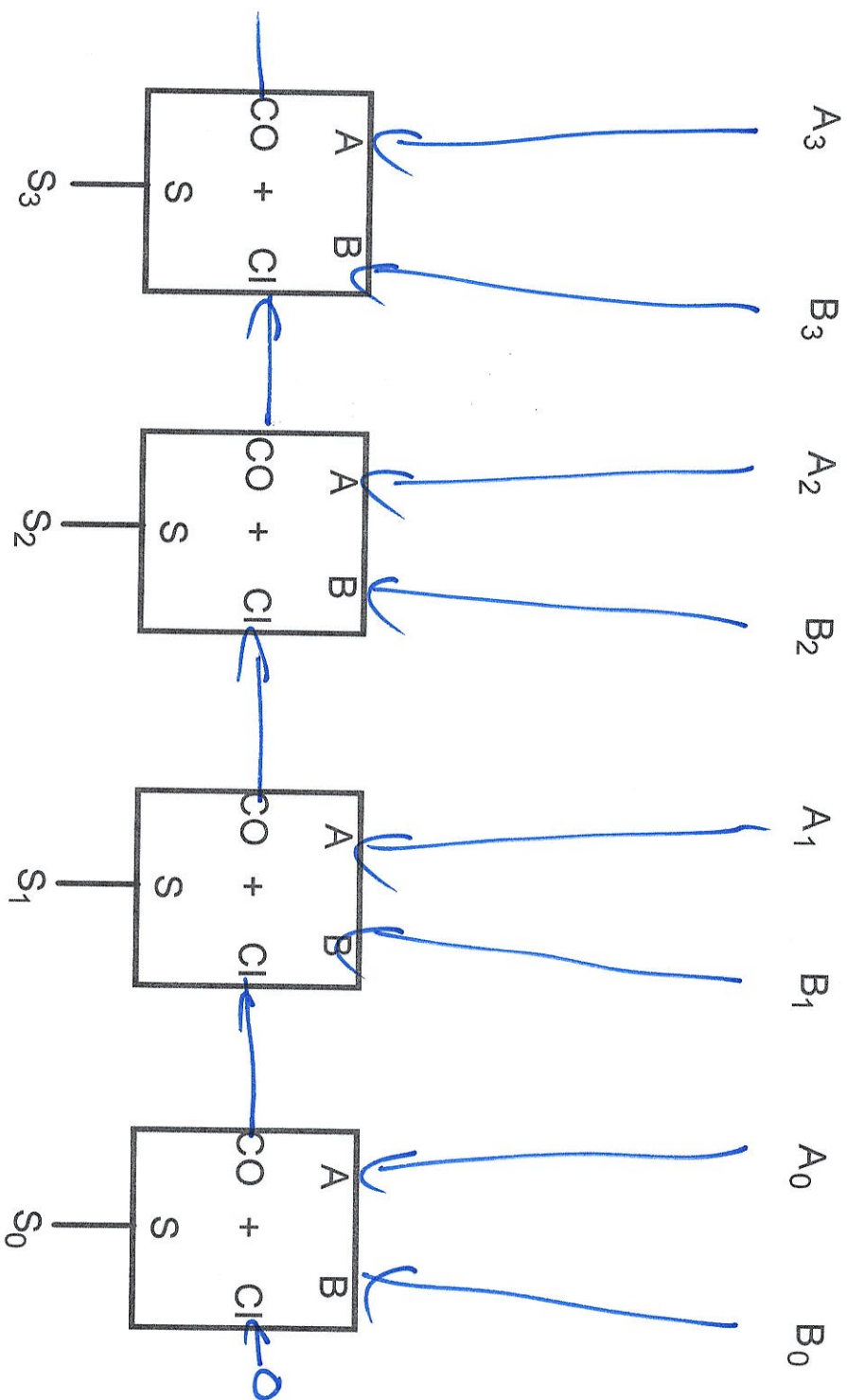


4xN ANDs



Multi-Bit Addition

$$\begin{array}{r}
 \text{Carry } C_3 \text{ } C_2 \text{ } C_1 \text{ } C_0 \\
 A_3 A_2 A_1 A_0 \\
 + B_3 B_2 B_1 B_0 \\
 \hline
 S_3 S_2 S_1 S_0
 \end{array}$$



Subtract

Adder/Subtractor

$$\begin{aligned} 0: A + B &= A + B + 0 \\ 1: A - B &= A + (\bar{B} + 1) \end{aligned}$$

$$X \oplus Y = X \oplus Y$$

