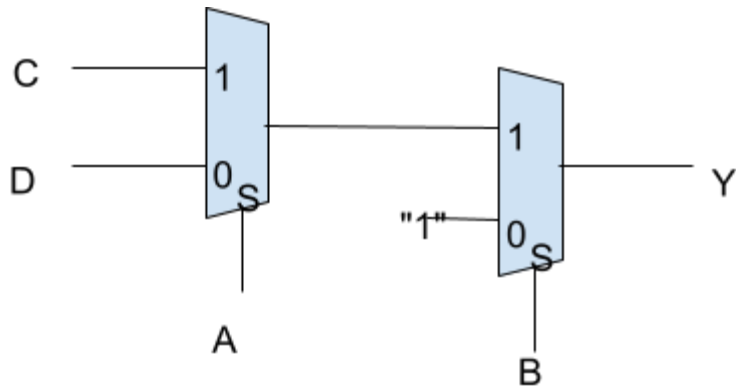


HW10: Digital Logic & Sequential Logic

Not graded

Question 1

Complete the truth table for the following circuit. (that is what is the value of Y for a given set of inputs).



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1

A	B	C	D	Y
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

To fill out the truth table, we enumerate all the inputs (A B C D) and write the output value for each combination. Remember a mux has the equation

$$\text{Out} = S ? \text{In1} : \text{In0}$$

We can tell from the circuit that the select input selects the input labelled "1" when $S=1$ and the input labelled "0" when $S=0$.

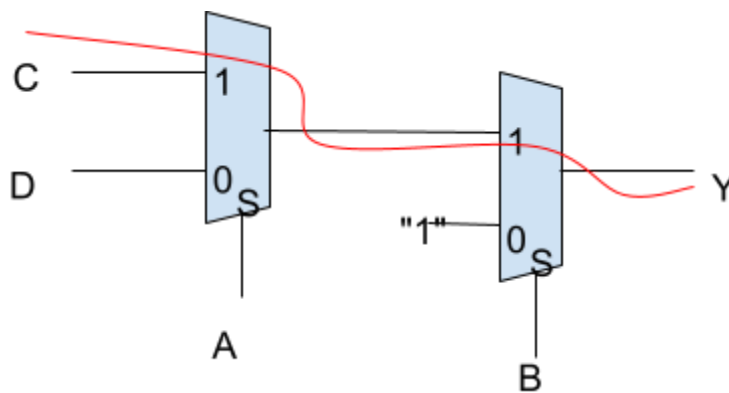
The mux on the right selects its bottom input which is "1" whenever B is 0. So we know that in the truth table, all entries with $B=0$ will have $Y=1$.

When B is 1, the rightmost mux selects the top input. This input comes from the mux on the left. The leftmost mux select is A. When $A=1$, the output of this mux is the value of C. When $A=0$, the output of this mux is the value of D.

For example, consider the row with A B C D

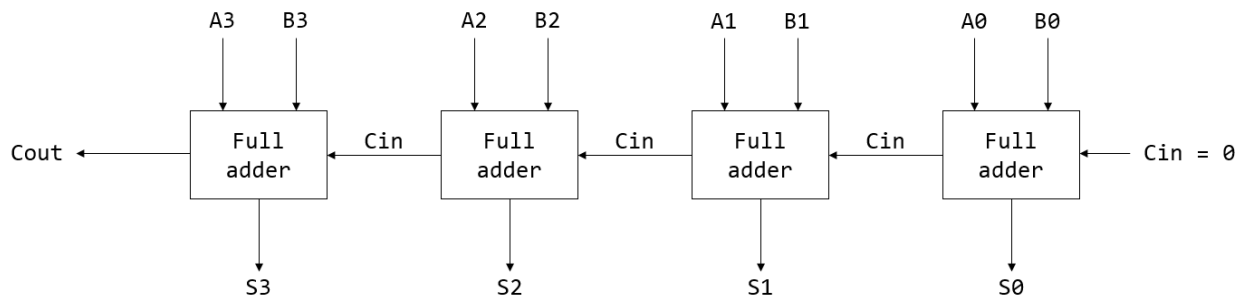
1 1 1 0

$B=1$ so the right most mux selects the output the left most mux. $A=1$ so the leftmost mux selects the value of C. $C=1$, so $Y = 1$.

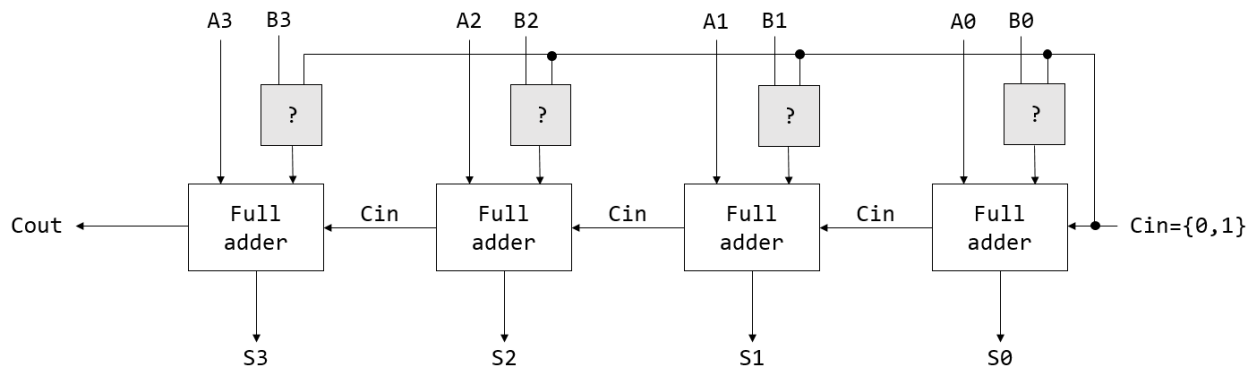







Question 2

Below we have a diagram of a 4-bit ripple carry adder.



What if we wanted to modify this to also handle subtraction? Instead of the initial carry in (C_{in}) value for the first bit adder being 0, we can imagine it being either a 1 or a 0. If $C_{in} = 0$, then our adder will perform addition, but if $C_{in} = 1$, then our adder will perform subtraction. What is the appropriate gate that belongs in the “?” boxes below to achieve this functionality?

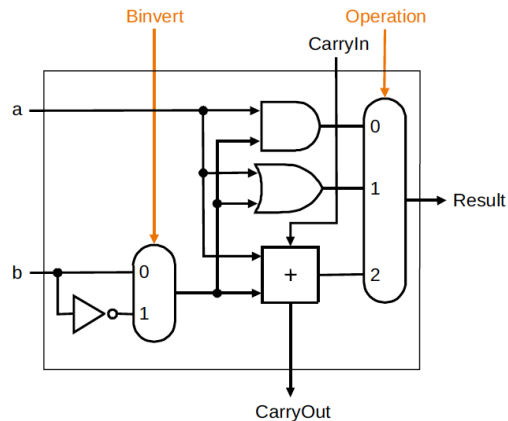


A. 	B. 	C. 
D. 	E. 	

To subtract, we need to add the 2's complement of B. That is, we need to flip all the bits of B and add 1. Setting $C_{in}=1$ will add 1. To flip all the bits, we use the XOR gate. Any signal X exclusive OR'd with "1" will flip the bit X.

Question 7 [3 points]: ALU (graded for correctness)

Given the following diagram of a 1-bit ALU, fill out the table below for what signals will configure the ALU for the given instruction. If a signal doesn't matter, indicate with an "X".



	Binvert	CarryIn	Operation(2-bits)
Add	0	0	10
Subtract	1	1	10
And	0	x	00
Or	0	x	01
BIC (Bit Clear)	1	x	00

The rightmost column controls the 3 input mux. We select the appropriate unit for each of Add/Subtract, And and OR and BIC.

Add - Operation = 10 to select the 2 input of the mux. CarryIn=0 and we don't want to invert B so Binvert = 0

Subtract - same as add but we set CarryIn=1 and we want to invert B (so we get A + (-B)). Where -B is the two's complement of B.

AND - we need to select the 0 input of the mux so Operation = 00. CarryIn doesn't matter so we set it to X. Binvert = 0 because we don't want to invert B.

Or - we need to select the 1 input of the mux so Operation = 01 CarryIn doesn't matter so we set it to X. Binvert = 0 because we don't want to invert B.

BIC - recall bit if clear is $A \& \sim B$. So we need to select the AND gate (Operation = 00). And we need to invert B so Binvert=1. Finally, CarryIn doesn't matter so we enter X in the table.

Question 8 [2 pts]: Machine Code

In our subset of the ARM instruction set, how many bits are used to specify the destination register of an add instruction

Answer	4
---------------	----------

There are 16 registers in our ARM architecture, so we need 4 bits in the instruction to specify one out of 16 registers.

For the ARM instruction data processing instructions, the machine code specifies fields Rd, Rn and Operand2. What value (in binary) would you have in field Rn. for the instruction add r2, r4, r6.

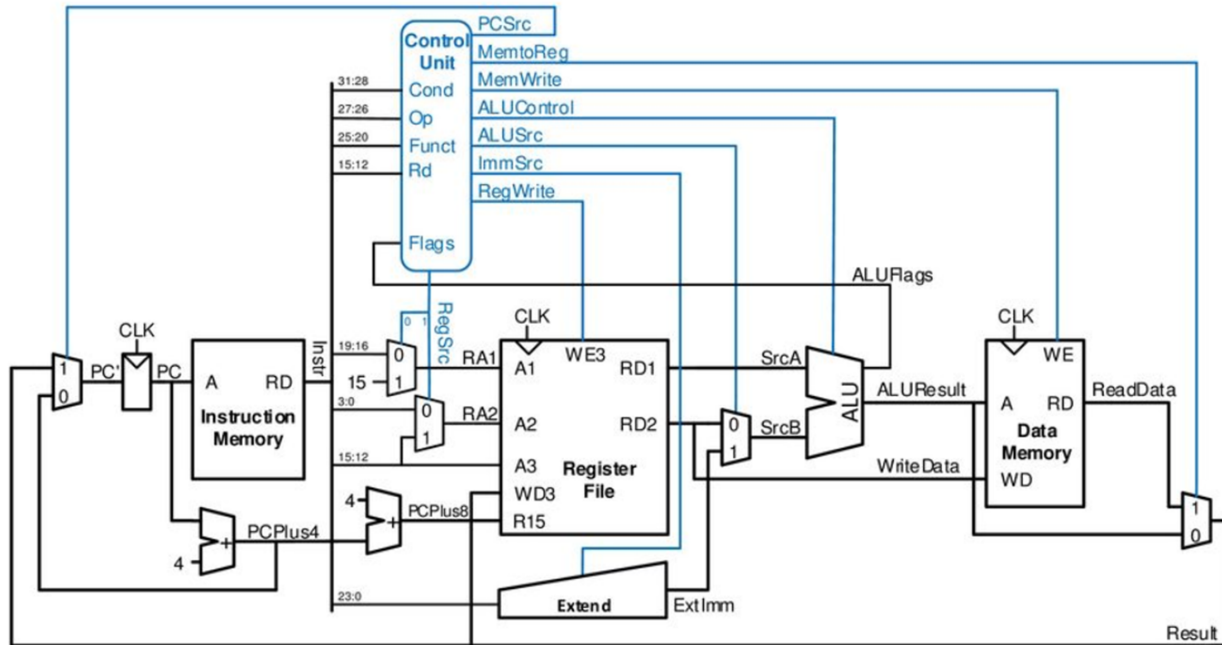
Answer	0100
---------------	-------------

This ARM instruction uses Rd - destination register, Rn - source registration, Operand2 -second source register. The instruction mnumonics are operand Rd, Rn, Operand2.

Rn = r4 so that field with be 0100

Question 9 [1 pts]: Datapath (graded for completion)

Given this single cycle datapath from lecture, answer the following questions below: (put X if the signal's value doesn't matter)



- A. What are the values on RegSrc, MemtoReg, MemWrite, ALUSrc, and RegWrite wire for the following instructions?

Rn Rd R2

a. add r3, r1, r2

RegSrc[1:0]	RA1[3:0]	RA2[3:0]	MemtoReg	MemWrite	ALUSrc	RegWrite
00	0001	0010	0	0	0	1

RegSrc = 00

RegSrc[0]=0 RA1 needs to have r1, not the value 15

RegSrc[1]=0 RA2 needs to have r2, which are in the low order bits of the instruction (bottom 4 bits of Operand2). **see arm reference sheets**

RA1 = 0001 because Rn is r1

RA2 = 0010 because Operand2 is r2

MemToReg = 0 because we want to select the "0" (bottom) input of the righthmost mux to allow the result from the ALU to be written back to the register file.

MemWrite =0 because we don't want to write data memory

ALUSrc = 0 because we want to use the output of the register file port RD2 so we can add $r1 + r2$.

RegWrite=1 because we want to write the register file.

b. str r2, [r3, #100] writing from register to memory (Not writing into any register)

RegSrc[1:0]	RA1[3:0]	RA2[3:0]	MemtoReg	MemWrite	ALUSrc	RegWrite
x0	0011	0010	x	1	1	0

c. ldr r3, [r4, #7]

RegSrc[1:0]	RA1[3:0]	RA2[3:0]	MemtoReg	MemWrite	ALUSrc	RegWrite
x0	0100	x	1	0	1	1

B. So far we have discussed add, sub, ldr, str, b instructions. Answer the following questions:

a. If the RegWrite wire was stuck on 0, which of the above instructions would **NOT** break?

Answer	str, b
--------	--------

b. If the RegSrc[0] wire were stuck at 1, which of the above instructions would break?

Answer	add, sub, ldr, str
--------	--------------------

- c. If the ALUSrc wire was stuck on 1, which of the instructions from part A would work correctly?

Answer	str and ldr
---------------	--------------------