CSE 232 SPRING 2020

PROJECT 2

ABDULLAH CELIK

171044002

Editing C Code:

We will implement following C code with our machine.

```
mult = 0;
while(a > 0){
        mult = mult + b;
        a = a - 1;
}
```

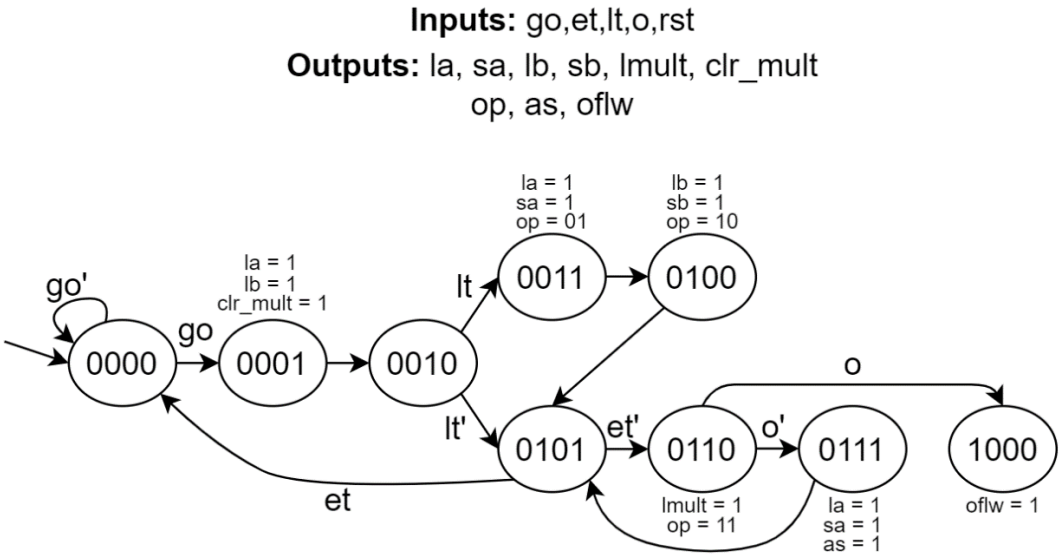
However, this C code is insufficient for negative numbers. Because I will also implement negative number multiplication. First, let's edit the code above.

```
if(a < 0){
        a = -a;
        b = -b;
}
mult = 0;
while(a > 0){
        mult = mult + b;
        a = a - 1;
}
```

State Diagram

To simplify the diagram below, outputs that it's 0 are not shown in the diagram. And also the rst input is not shown in the diagram. If rst input is 1 in any state, state will be returned 0000.

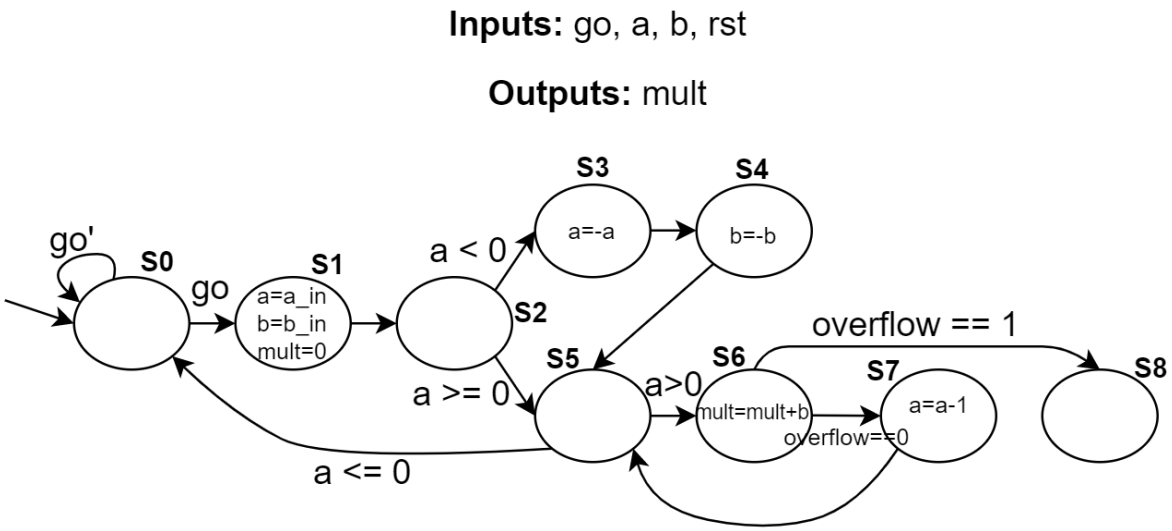
- go = run (input)
- et = equal than (input)
- lt = less than (input)
- o = overflow (input)
- rst = reset (input)
- la = load a (output)
- sa = select a (output)
- lb = load b (output)
- sb = select b (output)
- lmult = load mult (output)
- clr_mult = clear mult (output)
- op = operation (output)
- as = addition or substraction(output)
- oflw = overflow(output)



Draw Datapath

If rst input is 1 in any state, state will be returned S0 state. It is not drawn to simplify.

Inputs		Operation
a	b	
> 0	> 0	Nothing
> 0	< 0	Nothing
< 0	> 0	a = -a, b = -b
< 0	< 0	a = -a, b = -b



Draw Turth Table

Inputs								Outputs												
s3	s2	s1	s0	et	lt	go	o	n3	n2	n1	n0	la	sa	lb	sb	lmult	clr_mult	op	as	oflw
0	0	0	0	X	X	0	X	0	0	0	0	0	0	0	0	0	0	00	0	0
0	0	0	0	X	X	1	X	0	0	0	1	0	0	0	0	0	0	00	0	0
0	0	0	1	X	X	X	X	0	0	1	0	1	0	1	0	0	1	00	0	0
0	0	1	0	X	0	X	X	0	1	0	1	0	0	0	0	0	0	00	0	0
0	0	1	0	X	1	X	X	0	0	1	1	0	0	0	0	0	0	00	0	0
0	0	1	1	X	X	X	X	0	1	0	0	1	1	0	0	0	0	01	0	0
0	1	0	0	X	X	X	X	0	1	0	1	0	0	1	1	0	0	10	0	0
0	1	0	1	0	X	X	X	0	1	1	0	0	0	0	0	0	0	00	0	0
0	1	0	1	1	X	X	X	0	0	0	0	0	0	0	0	0	0	00	0	0
0	1	1	0	X	X	X	0	0	1	1	1	0	0	0	0	1	0	11	0	0
0	1	1	0	X	X	X	1	1	0	0	0	0	0	0	0	1	0	11	0	0
0	1	1	1	X	X	X	X	0	1	0	1	1	1	0	0	0	0	00	1	0
1	0	0	0	X	X	X	X	1	0	0	0	0	0	0	0	0	0	00	0	1

Derive Boolean Expressions From The Truth Table

- n3 - when n3 output is 1

s3	s2	s1	s0	et	lt	go	o
0	1	1	0	X	X	X	1
1	0	0	0	X	X	X	X

n3 = s3's2s1s0'o + s3s2's1's0'

- n2 - when n2 output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	1	0	X	0	X	X
0	0	1	1	X	X	X	X
0	1	0	0	X	X	X	X
0	1	0	1	0	X	X	X
0	1	1	0	X	X	X	0
0	1	1	1	X	X	X	X

n2 = s3's2's1s0'lt' + s3's2's1s0 + s3's2s1's0' + s3's2s1's0et' + s3's2s1s0'o' + s3's2s1s0

n2 = s3's2's1(s0'lt' + s0) + s3's2s1'(s0' + s0et') + s3's2s1(s0'o' + s0)

n2 = s3's2's1lt' + s3's2's1s0 + s3's2s1's0' + s3's2s1'et' + s3's2s1o' + s3's2s1s0

n2 = s3's2's1lt' + s3's1s0(s2' + s2) + s3's2s1's0' + s3's2s1'et' + s3's2s1o'

n2 = s3's2's1lt' + s3's1s0 + s3's2s1's0' + s3's2s1'et' + s3's2s1o'

- n1 - when n1 output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	0	1	X	X	X	X
0	0	1	0	X	1	X	X
0	1	0	1	0	X	X	X
0	1	1	0	X	X	X	0

n1 = s3's2's1's0 + s3's2's1s0'lt + s3's2s1's0et' + s3's2s1s0'o'

n1 = s3's1's0(s2' + s2et') + s3's2's1s0'lt + s3's2s1s0'o'

n1 = s3's2's1's0 + s3's1's0et' + s3's2's1s0'lt + s3's2s1s0'o'

- n0 - when n0 output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	0	0	X	X	1	X
0	0	1	0	X	0	X	X
0	0	1	0	X	1	X	X
0	1	0	0	X	X	X	X
0	1	1	0	X	X	X	0
0	1	1	1	X	X	X	X

n0 = s3’s2’s1’s0’go + s3’s2’s1s0’lt’ + s3’s2’s1s0’lt + s3’s2s1’s0’ + s3’s2s1s0’o’ + s3’s2s1s0
n0 = s3’s1’s0’(s2’go + s2) + s3’s2’s1s0’(lt’ + lt) + s3’s2s1s0’o’ + s3’s2s1s0
n0 = s3’s1’s0’go + s3’s2s1’s0’ + s3’s2’s1s0’ + s3’s2s1s0’o’ + s3’s2s1s0
n0 = s3’s1’s0’go + s3’s2s1’s0’ + s3’s1s0’(s2’ + s2o’) + s3’s2s1s0
n0 = s3’s1’s0’go + s3’s2s1’s0’ + s3’s2’s1s0’ + s3’s1s0’o’ + s3’s2s1s0

- la - when la output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	0	1	X	X	X	X
0	0	1	1	X	X	X	X
0	1	1	1	X	X	X	X

la = s3’s2’s1’s0 + s3’s2’s1s0 + s3’s2s1s0
la = s3’s2’s0(s1’ + s1) + s3’s2s1s0
la = s3’s2’s0 + s3’s2s1s0
la = s3’s0(s2’ + s2s1)
la = s3’s0(s2’ + s1)

- sa - when sa output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	1	1	X	X	X	X
0	1	1	1	X	X	X	X

sa = s3’s2’s1s0 + s3’s2s1s0
sa = s3’s1s0(s2’ + s2)
sa = s3’s1s0

- lb - when lb output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	0	1	X	X	X	X
0	1	0	0	X	X	X	X

lb = s3’s2’s1’s0 + s3’s2s1’s0’
lb = s3’s1’(s2’s0 + s2s0’)
lb = s3’s1’(s2 XOR s0)

- sb - when sb output is 1

s3	s2	s1	s0	et	lt	go	o
0	1	0	0	X	X	X	X

sb = s3’s2s1’s0

- lmult - when lmult output is 1

s3	s2	s1	s0	et	lt	go	o
0	1	1	0	X	X	X	0
0	1	1	0	X	X	X	1

lmult = s3's2s1s0'o' + s3's2s1s0'o
lmult = s3's2s1s0'(o' + o)
lmult = s3's2s1s0'

- clr_mult - when clr_mult output is 1

s3	s2	s1	s0	et	lt	go	o
0	0	0	1	X	X	X	X

clr_mult = s3's2's1's0

- op - when any of the bits of op output are 1

0.bit

s3	s2	s1	s0	et	lt	go	o
0	0	1	1	X	X	X	X
0	1	1	0	X	X	X	0
0	1	1	0	X	X	X	1

op 0.bit = s3's2's1s0 + s3's2s1s0'o' + s3's2s1s0'o
op 0.bit = s3's2's1s0 + s3's2s1s0'(o' + o)
op 0.bit = s3's2's1s0 + s3's2s1s0'
op 0.bit = s3's1(s2's0 + s2s0')
op 0.bit = s3's1(s2 XOR s0)

1.bit

s3	s2	s1	s0	et	lt	go	o
0	1	0	0	X	X	X	X
0	1	1	0	X	X	X	0
0	1	1	0	X	X	X	1

op 1.bit = s3's2s1's0' + s3's2s1s0'o' + s3's2s1s0'o
op 1.bit = s3's2s1's0' + s3's2s1s0'(o' + o)
op 1.bit = s3's2s1's0' + s3's2s1s0'
op 1.bit = s3's2s0'(s1' + s1)
op 1.bit = s3's2s0'

- as - when as output is 1

s3	s2	s1	s0	et	lt	go	o
0	1	1	1	X	X	X	X

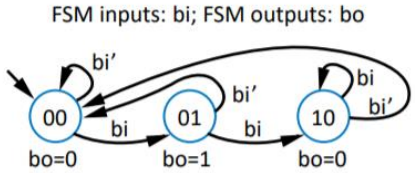
as = s3's2s1s0

- oflw - when oflw output is 1

s3	s2	s1	s0	et	lt	go	o
1	0	0	0	X	X	X	X

oflw = s3s2’s1’s0’

Button Press Synchronizer State Diagram

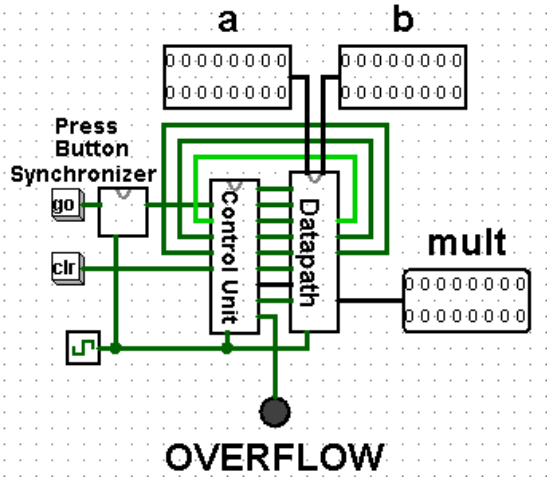


Button Press Synchronizer Truth Table and Boolean Expresions

Combinational logic					
Inputs			Outputs		
s1	s0	bi	n1	n0	bo
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

n1 = s1’s0bi + s1s0’bi
n0 = s1’s0’bi
bo = s1’s0bi’ + s1’s0bi = s1’s0(bi’ + bi) = s1’s0

Overview



Optimization

In some parts, I did not take any parenthesis because the critical path will increase, so I reduced the critical paths. Apart from that, I created an overflow state to prevent the process from continuing when there was overflow and prevented the process from continuing unnecessarily. I added a button press synchronizer to the go button of the user and no matter how long the user held it, I made the control unit work once.