CS151B Winter 2022

Discussion (Week 3)

CS151B Teaching Team

Agenda

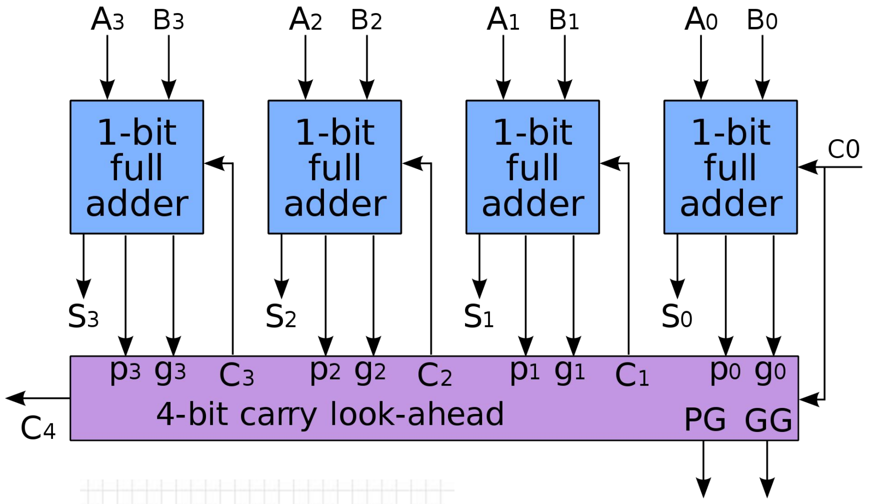
- Logistics
- Review of adders
- HCLA (Hierarchical Carry Look Ahead)
- Multiplication
- Booth’s algorithm
- Practice Questions
- Q&A

Logistics

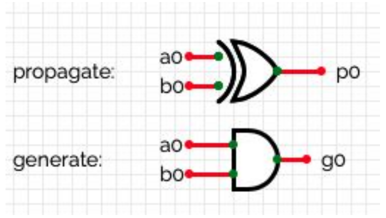
- Homework#3 is due **tonight 11:59pm, PST**.
- Midterm is two weeks away (Wednesday of Week 5)

Review

Carry-Lookahead Adder (CLA)



A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0
1-bit full adder
C_0
S_3 S_2 S_1 S_0
p_3 g_3 C_3 p_2 g_2 C_2 p_1 g_1 C_1 p_0 g_0
4-bit carry look-ahead
PG GG
C_4



pi = Ai XOR Bi
gi = Ai AND Bi

c0: ready at the beginning of the computation.

c1 = g0 + c0p0

c2 = g1 + g0p1 + c0p1p0

c3 = g2 + g1p2 + g0p1p2 + c0p0p1p2

c4 = g3 + g2p3 + g1p2p3 + g0p1p2p3 +c0p0p1p2p3

Q:How to calculate s4?

a4^b4^c4   AKA p4^c4

Image source: https://en.wikipedia.org/wiki/Carry-lookahead_adder#/media/File:4-bit_carry_lookahead_adder.svg

Carry Lookahead Adder (example)

- Calculate C4 using the pattern from C1-C3.

  C4 = G3 + G2*P3 + G1*P2*P3 + G0*P1*P2*P3 + C0*P0*P1*P2*P3

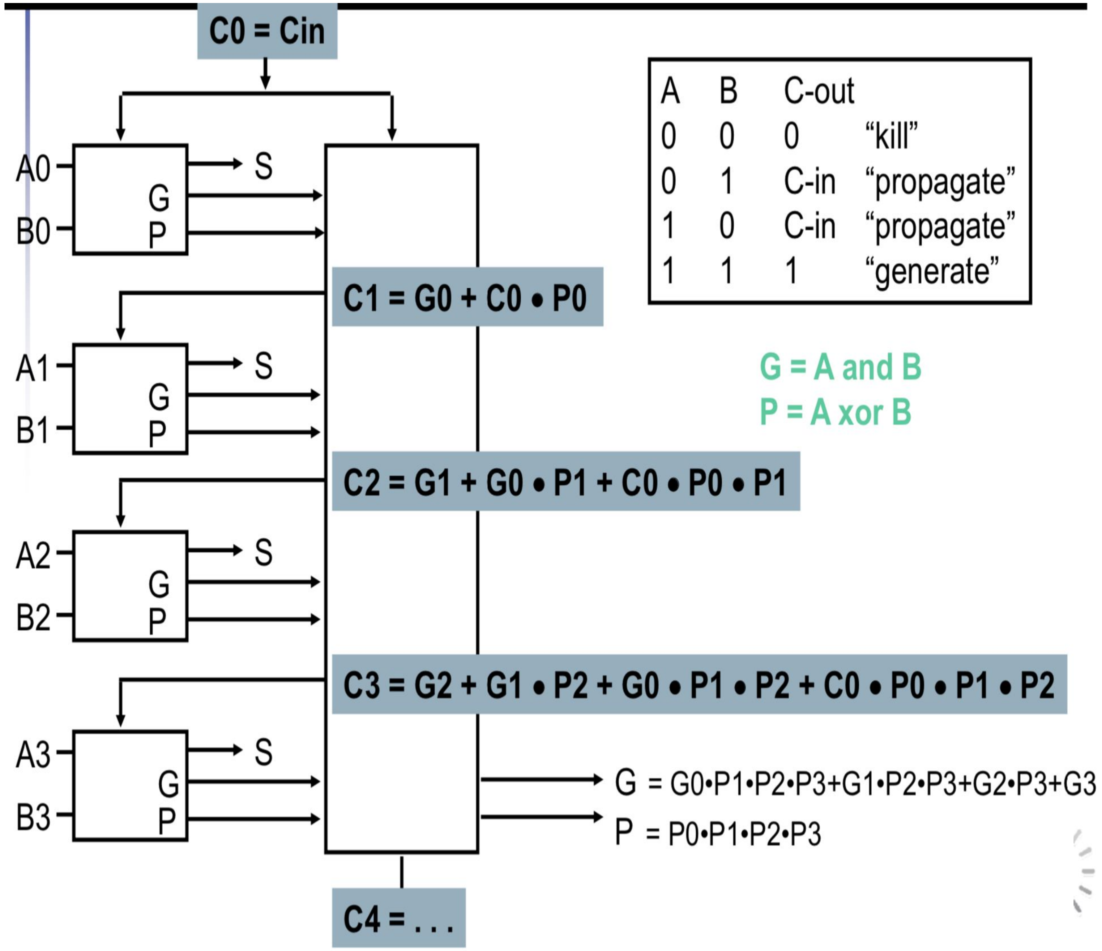
Note: We can treat C0 as G-1 for simpler reasoning

- What about C5?

  C5 = G4 + G3*P4 + G2*P3*P4 + G1*P2*P3*P4 + G0*P1*P2*P3*P4 + C0*P0*P1*P2*P3*P4

What's the problem with this design?

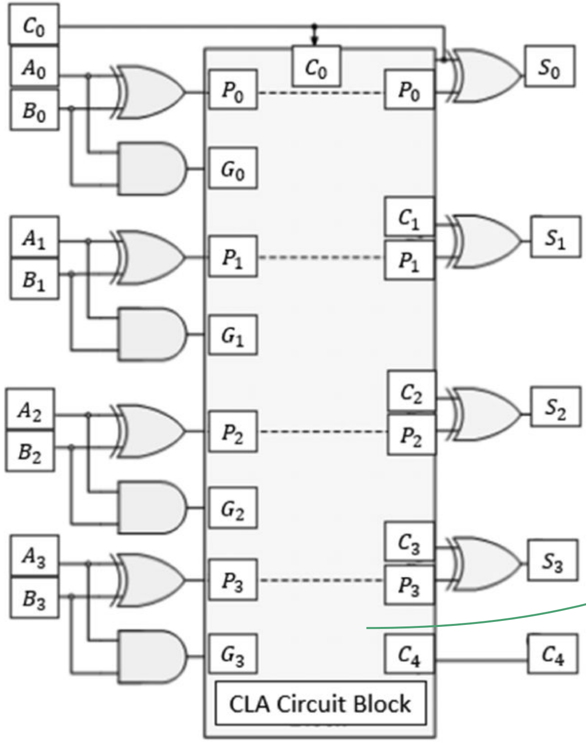
A: Equations can get quite long as we progress to the higher order bits



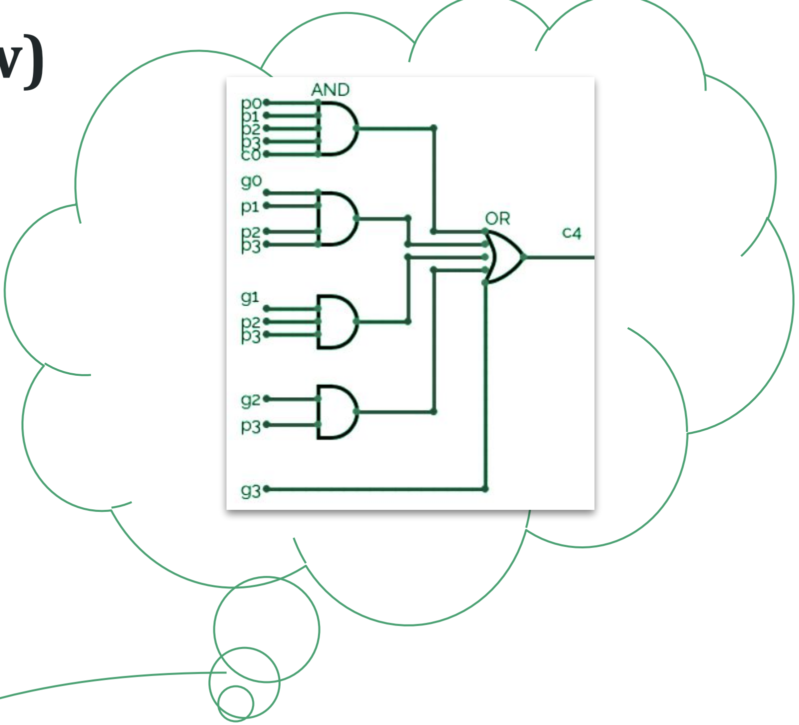
C0 = Cin
A0
B0
A1
B1
A2
B2
A3
B3
G
P
S
C1 = G0 + C0 • P0
C2 = G1 + G0 • P1 + C0 • P0 • P1
C3 = G2 + G1 • P2 + G0 • P1 • P2 + C0 • P0 • P1 • P2
C4 = . . .
G = G0•P1•P2•P3+G1•P2•P3+G2•P3+G3
P = P0•P1•P2•P3
G = A and B
P = A xor B

A	B	C-out	
0	0	0	"kill"
0	1	C-in	"propagate"
1	0	C-in	"propagate"
1	1	1	"generate"

4-bit Carry Lookahead Adder(overview)



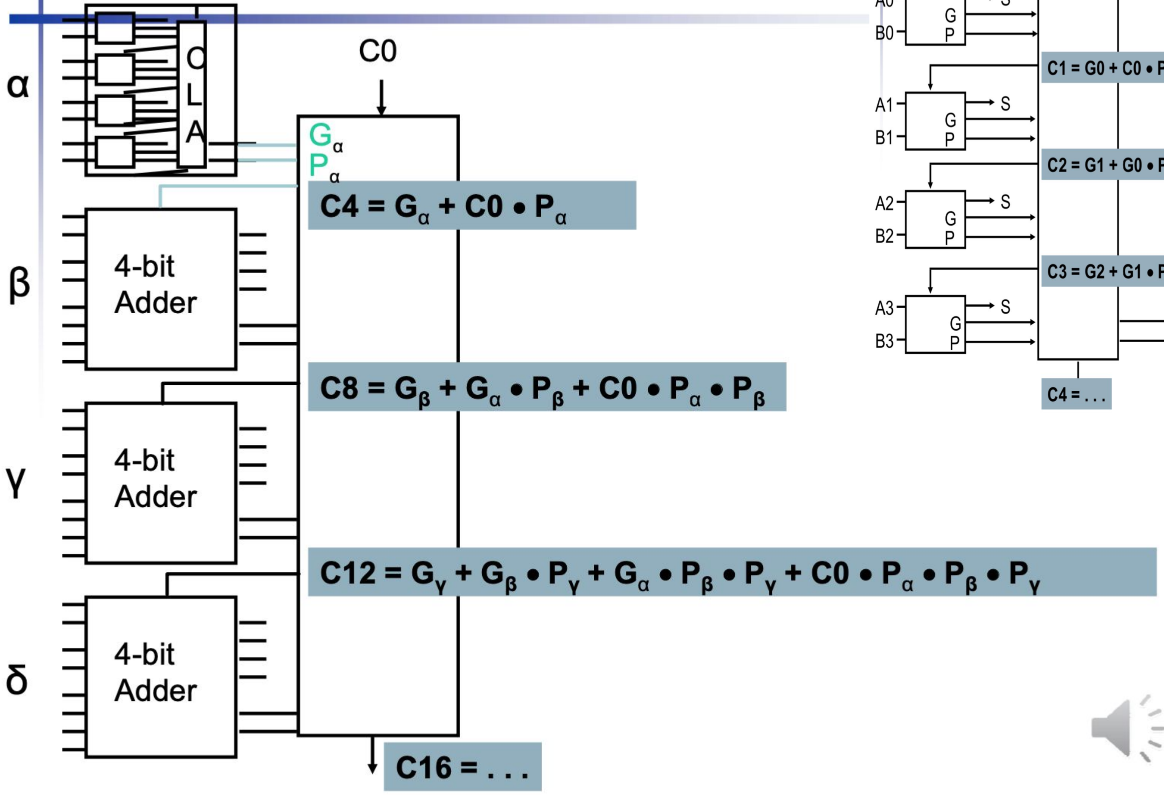
g &p logic Carry logic



Q: Is this efficient ?

Q: Is this efficient ?

Hierarchical CLA



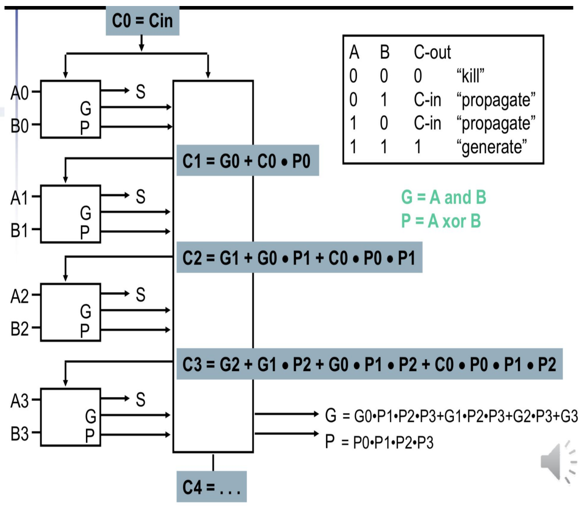
G_α, P_α

$$C4 = G_\alpha + C0 \bullet P_\alpha$$

$$C8 = G_\beta + G_\alpha \bullet P_\beta + C0 \bullet P_\alpha \bullet P_\beta$$

$$C12 = G_\gamma + G_\beta \bullet P_\gamma + G_\alpha \bullet P_\beta \bullet P_\gamma + C0 \bullet P_\alpha \bullet P_\beta \bullet P_\gamma$$

$$C16 = \ldots$$



$C0 = Cin$

$C1 = G0 + C0 \bullet P0$

$C2 = G1 + G0 \bullet P1 + C0 \bullet P0 \bullet P1$

$C3 = G2 + G1 \bullet P2 + G0 \bullet P1 \bullet P2 + C0 \bullet P0 \bullet P1 \bullet P2$

$C4 = \ldots$

$G = G0 \cdot P1 \cdot P2 \cdot P3 + G1 \cdot P2 \cdot P3 + G2 \cdot P3 + G3$

$P = P0 \cdot P1 \cdot P2 \cdot P3$

A	B	C-out	
0	0	0	"kill"
0	1	C-in	"propagate"
1	0	C-in	"propagate"
1	1	1	"generate"

G = A and B
P = A xor B

Q: How do we calculate C15?

C15 = G14 + G13*P14 + G0*P13*P14 + C12*P12*P13*P14

Q: Which is faster to calculate C15 vs C16?

C16

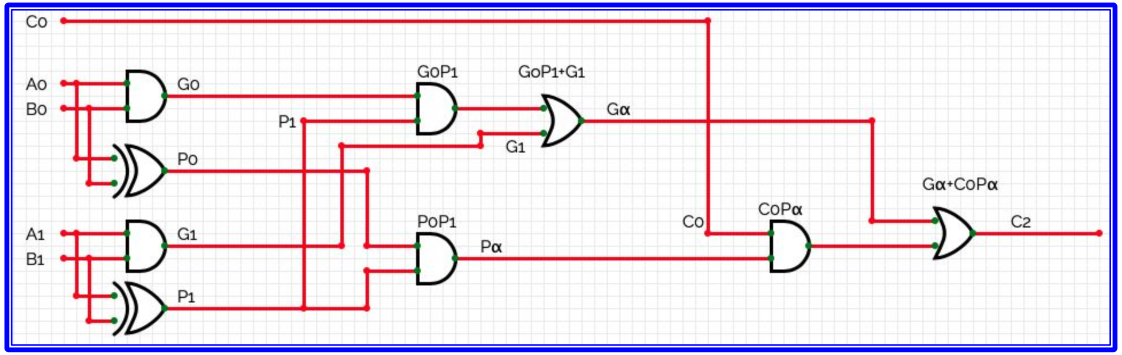
Q: How do we calculate S15?

(A15^B15)^C15 AKA P15^C15

4-bit HCLA (2 stack of *2 bits CLA*)

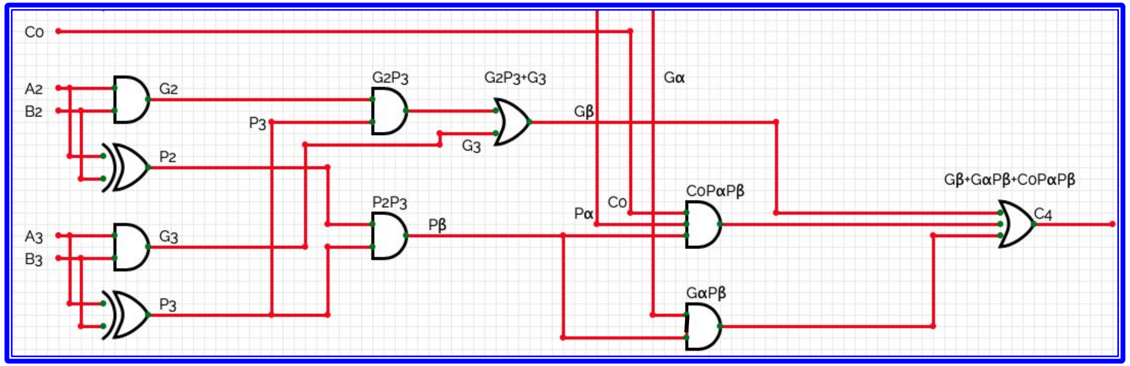
α-block

Gα = G0P1+G1
Pα = P0P1
C2 = Gα + C0Pα



β-block

Gβ = G2P3+G3
Pβ = P2P3
C4 = Gβ + GαPβ+C0PαPβ

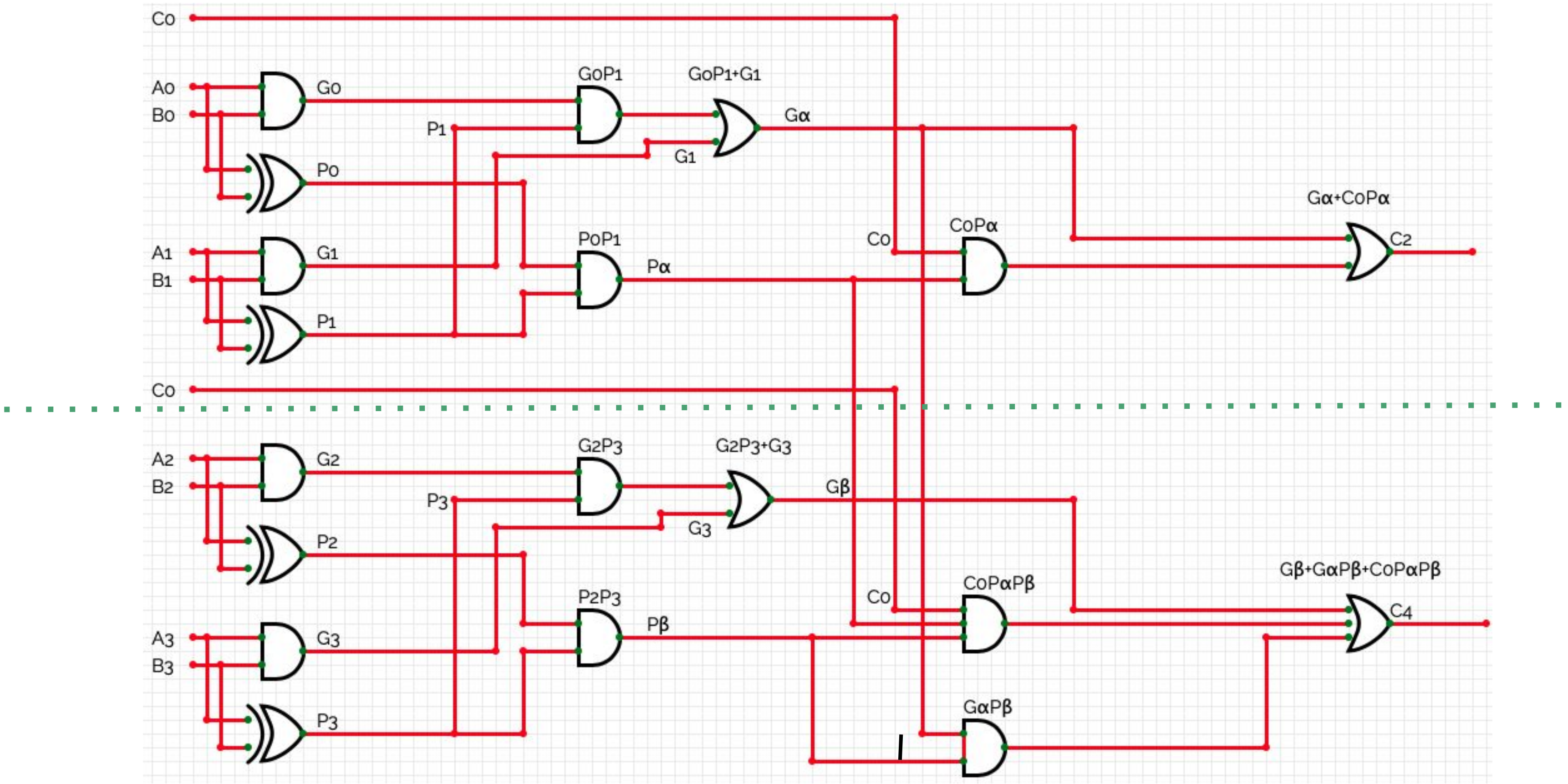


Q: How do we find C1, C3 ?

A: Use the same formula we use for carry look ahead logic:

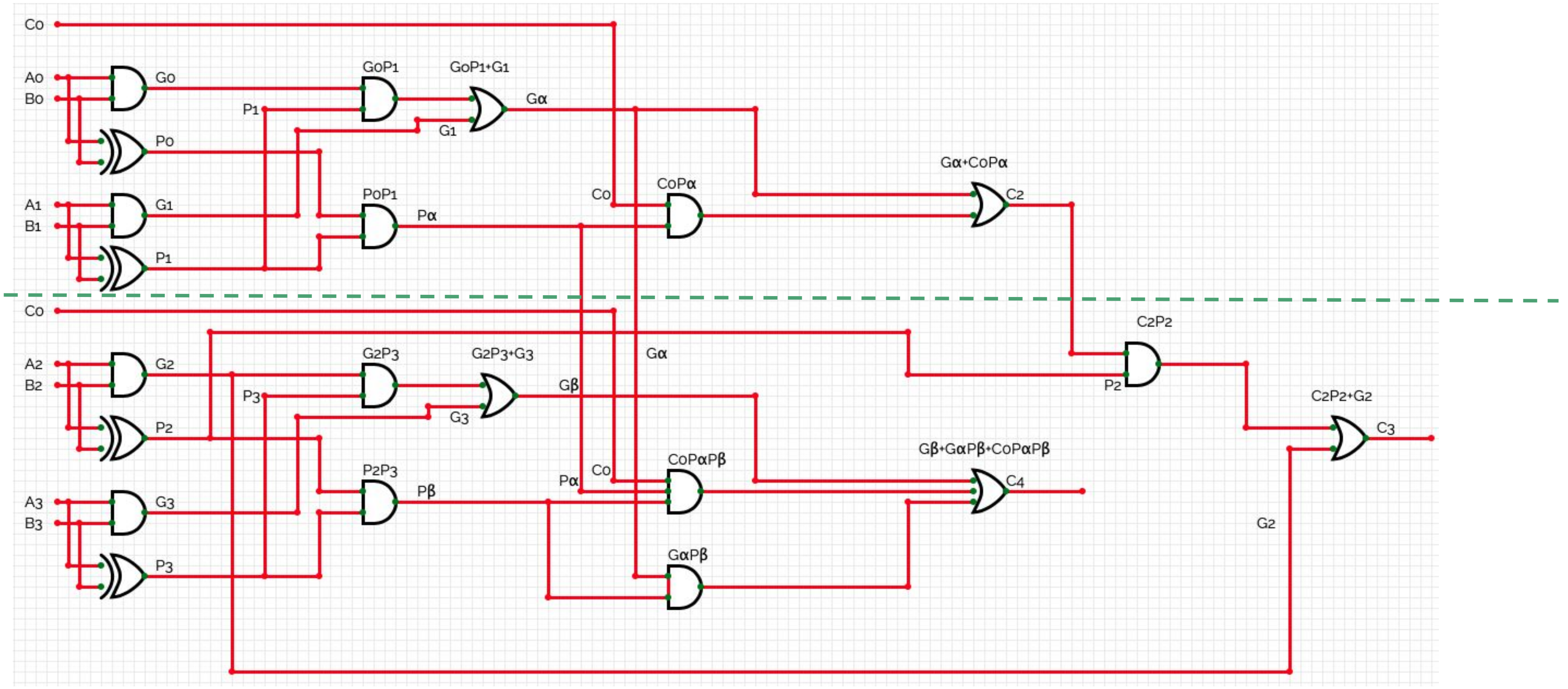
C1=C0P0 + G0
C3=C2P2 + G2

4-bit HCLA (*Full View*)

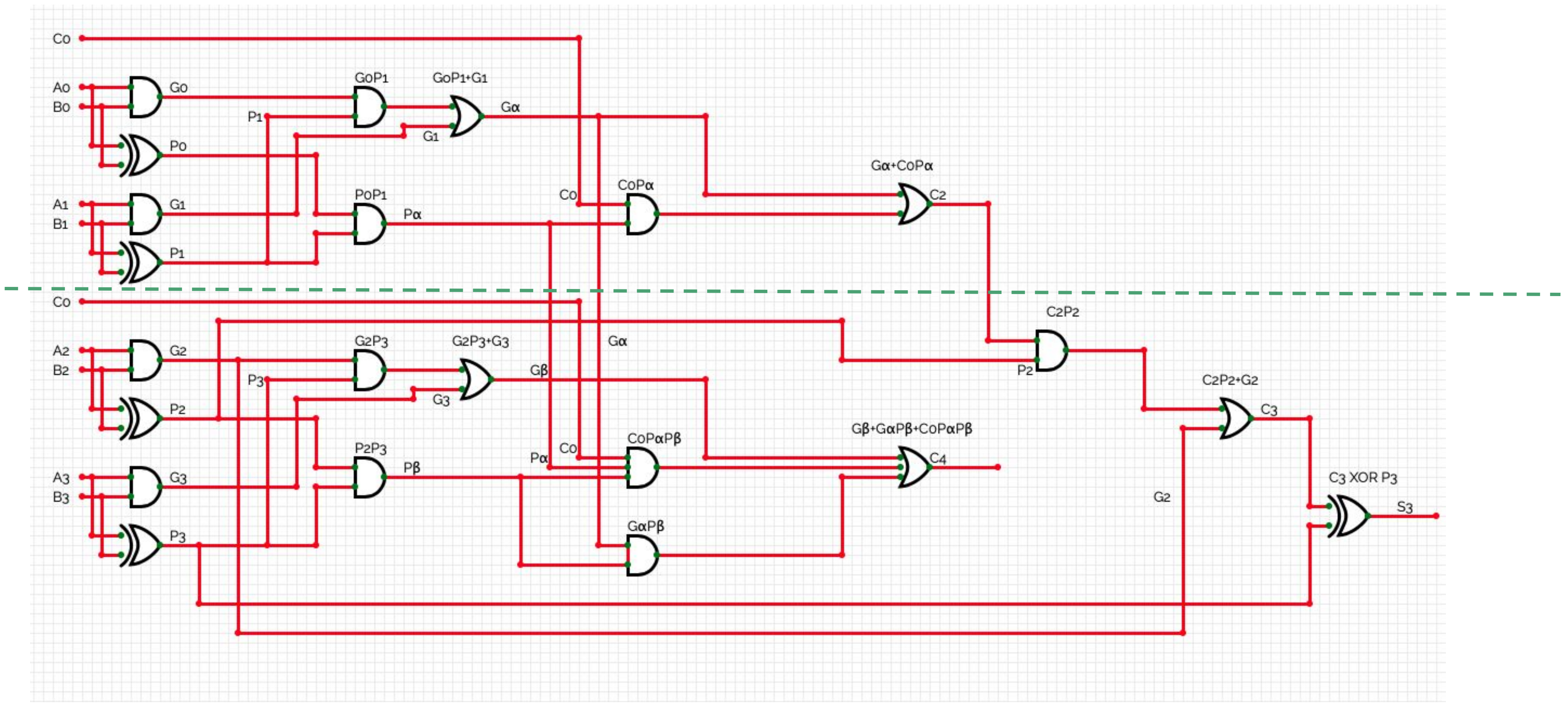


Of course, we still need to implement the logic for the rest of the carry bits and the sum bits, in a few trivial steps.
Note that C2 will be used as a carry in for the entire β block, to further calculate other carryout bits.

HCLA - C3 calculation



HCLA - S3 calculation



Multiplication

How do we do multiplication in binary ?

(Binary) Multiplication by left-shifting multiplicand

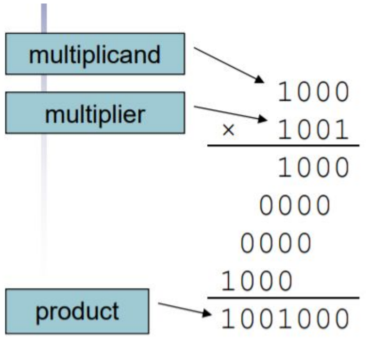
Note we sum left-shifted versions of multiplicand (md)

Depends on multiplier bit (control)

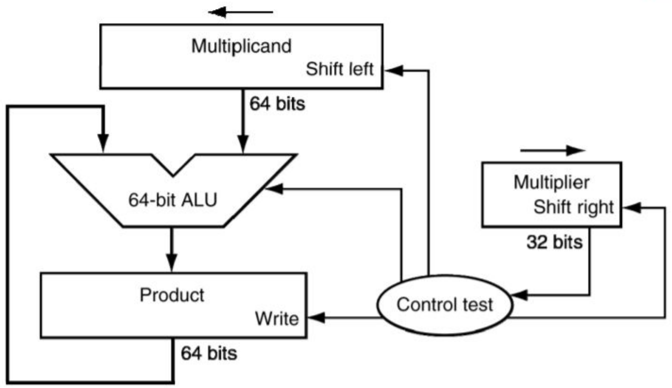
For each bit in multiplier:

    Sum += md*bit

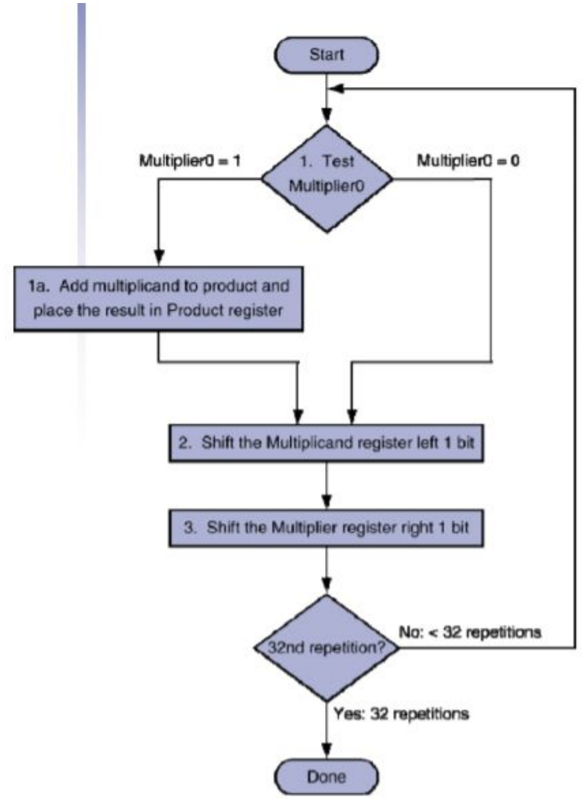
    md left shift

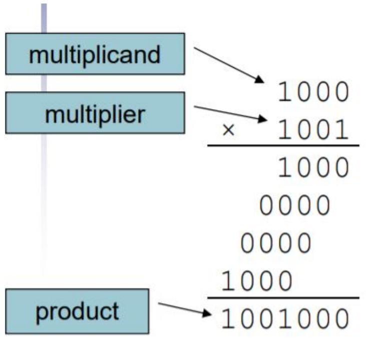


Length of product is the sum of operand lengths

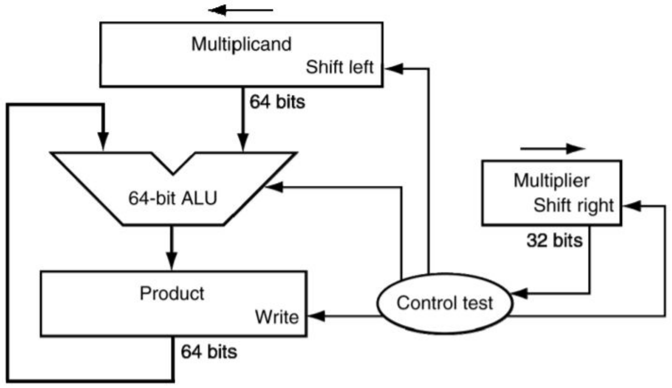


(Binary) Multiplication by left-shifting multiplicand


Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done



Length of product is the sum of operand lengths



Is the state diagram clear?

(Binary) Multiplication optimization

#1: Use 32-bit ALU. Why?

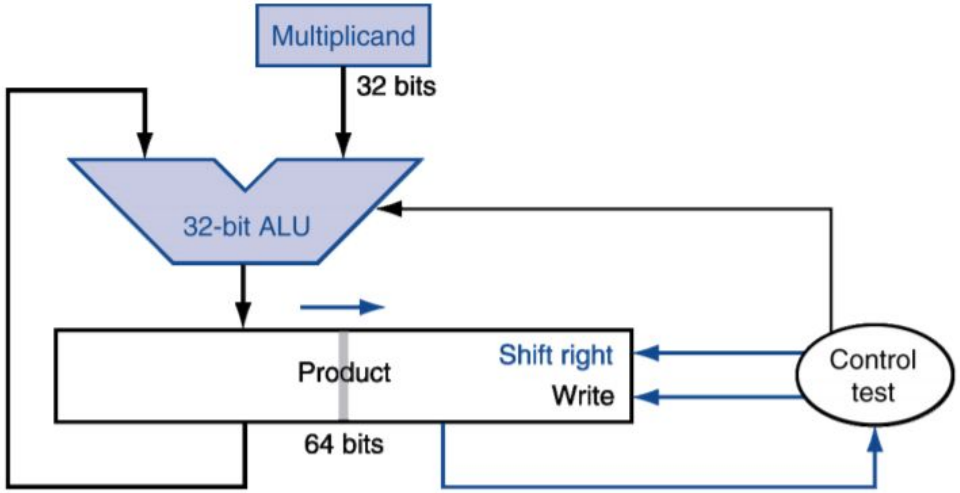
#2: Right-shift product. Why?

Example:
Multiplicand = 100
Multiplier = 101

LSB of multiplier is 1, so product is: 100 101
Right shift: product = 010 010

Next bit is 0, product remains 010 010
Right shift: product = 001 001

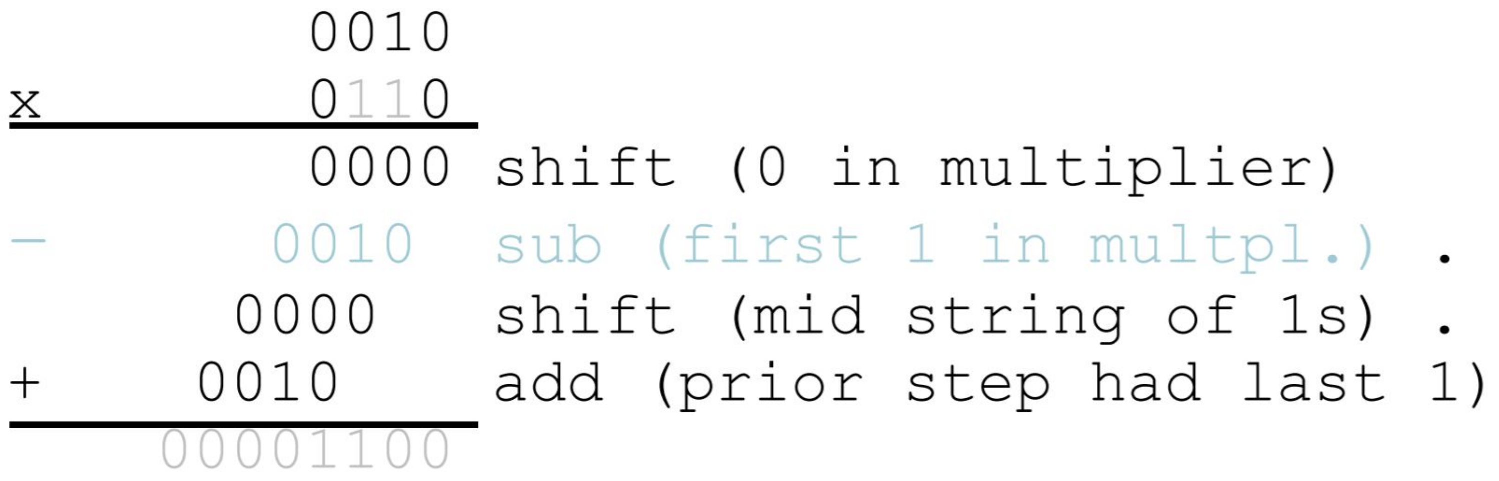
Last, bit is 1 so product = 101 001
Right shift: product = 010 100 ⇒ Done



(Binary) Multiplication Booth's Algorithm

Idea: 1 + 2 + 4 + 8 + ... + 2^n = 2^(n+1) - 1
Leverage this fact to change (n+1) adds into 2 adds

0011 1110 = ? (in this case it's pretty efficient)
1001 1011 = ? (in this case it's not as efficient)



```
            0010
x           0110
----------------
            0000  shift (0 in multiplier)
-          0010   sub (first 1 in multpl.)  .
          0000    shift (mid string of 1s)  .
+        0010     add (prior step had last 1)
----------------
        00001100
```

Booth's algorithm - example(1)

```
5*3 = 15

    0101
    0011(0)
```

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

Booth's algorithm - example(1)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
5*3 = 15

    0101
    0011(0)
-----------------
   -0101
```

Multiplier bits checked: 10 (subtract)

Booth's algorithm - example(1)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
5*3 = 15

    0101
    0011(0)
-----------------
   -0101
   0000   Multiplier bits checked: 11 (shift)
```

Booth's algorithm - example(1)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
5*3 = 15

      0101
      0011(0)
  ________________
     -0101
     0000
    +0101   Multiplier bits checked: 01 (add)
```

Booth's algorithm - example(1)

```
5*3 = 15

    0101
    0011(0)
-----------------
   -0101
   0000
 +0101
 0000
```

Multiplier bits checked: 00 (shift)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

Booth’s algorithm - example(1)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
5*3 = 15

    0101
    0011(0)
---------------------
   -0101                    1111 1011
   0000                        0 000
  +0101             =         01 01
  0000                       000 0
                           ---------------------

                            0000 1111

                            =15
```

Booth’s algorithm - example(2)

```
7*(-3) = -21

    0111
    1101(0)
---------------
```

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

Booth's algorithm - example(2)

key:
00 = shift
11 = shift
10 = subtract
01 = add

```
7*(-3) = -21

    0111
    1101(0)
------------------
   -0111      Multiplier bits checked: 10 (subtract)
```

Booth’s algorithm - example(2)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
7*(-3) = -21

    0111
    1101(0)
-----------------
   -0111
  +0111   Multiplier bits checked: 01 (add)
```

Booth’s algorithm - example(2)

```
7*(-3) = -21

    0111
    1101(0)
----------------
   -0111
  +0111
 -0111    Multiplier bits checked: 10 (subtract)
```

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

Booth’s algorithm - example(2)

key:

00	=	shift
11	=	shift
10	=	subtract
01	=	add

```
7*(-3) = -21

    0111
    1101(0)
________________
   -0111
  +0111
 -0111
 0000     Multiplier bits checked: 11 (shift)
```

Booth’s algorithm - example(2)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
7*(-3) = -21

    0111
    1101(0)
________________
   -0111
  +0111
 -0111
 0000
```

=

```
1111 1001
   0 111
1110 01
 000 0
__________
1110 1011
```

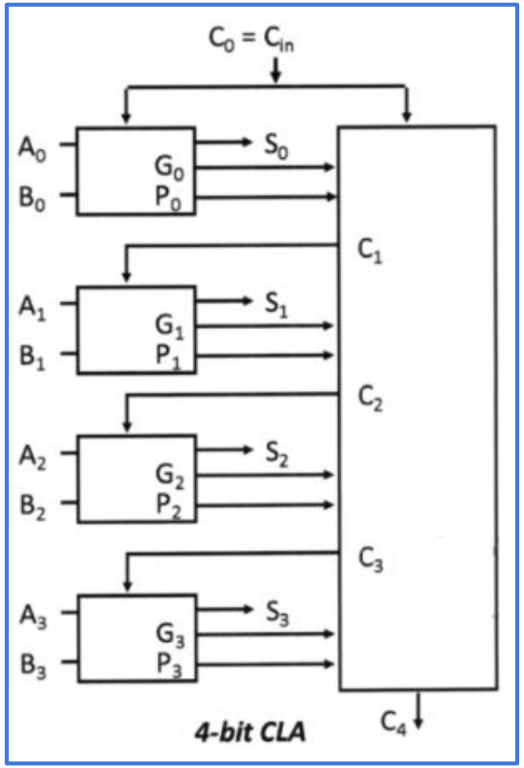
= -21

Practice Questions

One Clarification

P5 (4-bit CLA)

How much optimization??? Leaves have to be done in parallel.



Fan in	Delays
2	4T
3	6T
4	9T
5	13T
6	17T

result table

output	delay
C3	22T
S3	26T
C4	30T

c3 = g2 + p2g1 + p2p1g0 + p2p1p0c0
4T(getting g,p)+9T(4inputand)+9T(4inputor) = 22T

s3 = (a2 xor b2) xor c3 //a2 and b2 can be precomputed
c3 = 22T
s3 <- 22T+4T(2inputxor) = 26T

c4 = g3 + g2p3 + g1p2p3 + g0p1p2p2 +c0p0p1p2p3
4T(getting g,p)+13T(5inputand)+13T(5inputor) = 30T

This question was collected from a previous midterm exam

Practice Questions

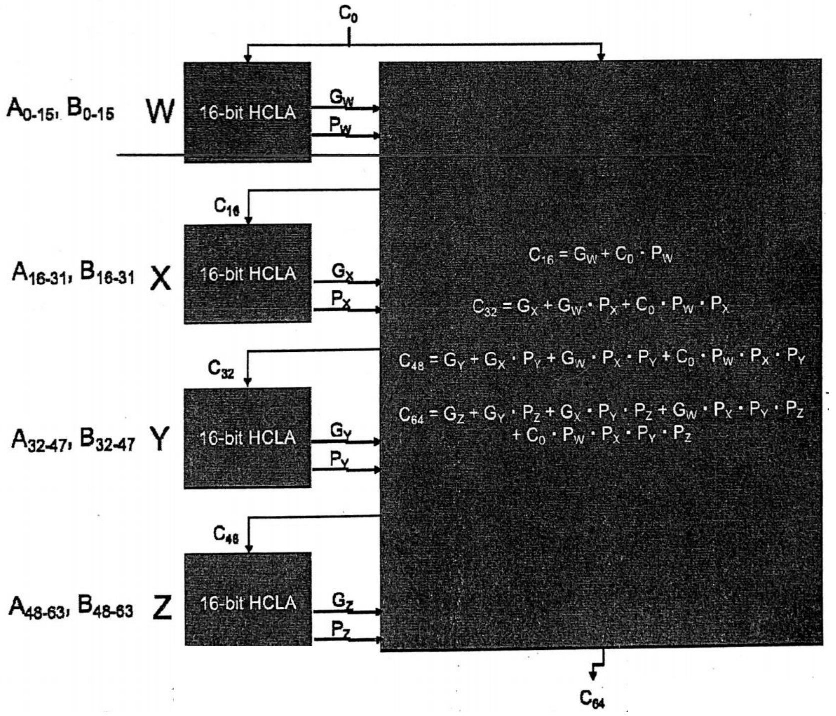
P1 - HCLA

1) Find the equations for G_w and P_w in terms of $G_\alpha, G_\beta, G_\gamma, G_\delta, P_\alpha, P_\beta, P_\gamma, P_\delta$

$G_W = G_\alpha P_\beta P_\gamma P_\delta + G_\beta P_\gamma P_\delta + G_\gamma P_\delta + G_\delta$

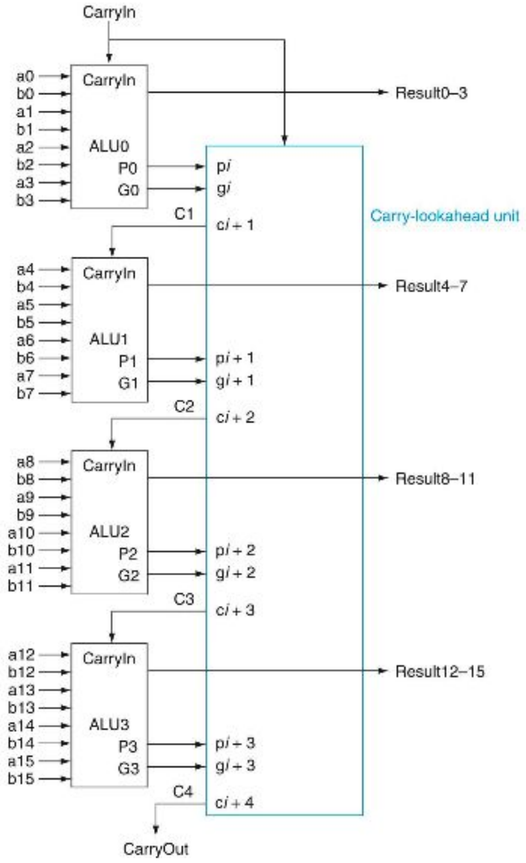
$P_W = P_\alpha P_\beta P_\gamma P_\delta$

fan-in	AND/ OR	XOR
2	1T	2T
3	2T	3T
4	5T	6T
5	7T	9T
6	10T	12T
7	13T	16T
8	15T	18T



C_0
A_{0-15}, B_{0-15} W
16-bit HCLA
G_W
P_W
C_{16}
A_{16-31}, B_{16-31} X
16-bit HCLA
G_X
P_X
C_{32}
A_{32-47}, B_{32-47} Y
16-bit HCLA
G_Y
P_Y
C_{48}
A_{48-63}, B_{48-63} Z
16-bit HCLA
G_Z
P_Z
$C_{16} = G_W + C_0 \cdot P_W$
$C_{32} = G_X + G_W \cdot P_X + C_0 \cdot P_W \cdot P_X$
$C_{48} = G_Y + G_X \cdot P_Y + G_W \cdot P_X \cdot P_Y + C_0 \cdot P_W \cdot P_X \cdot P_Y$
$C_{64} = G_Z + G_Y \cdot P_Z + G_X \cdot P_Y \cdot P_Z + G_W \cdot P_X \cdot P_Y \cdot P_Z + C_0 \cdot P_W \cdot P_X \cdot P_Y \cdot P_Z$
C_{64}

This question was collected from a previous midterm exam


CarryIn
a0
b0
a1
b1
a2
b2
a3
b3
CarryIn
ALU0
P0
G0
Result0–3
pi
gi
C1
ci + 1
Carry-lookahead unit
a4
b4
a5
b5
a6
b6
a7
b7
CarryIn
ALU1
P1
G1
Result4–7
pi + 1
gi + 1
C2
ci + 2
a8
b8
a9
b9
a10
b10
a11
b11
CarryIn
ALU2
P2
G2
Result8–11
pi + 2
gi + 2
C3
ci + 3
a12
b12
a13
b13
a14
b14
a15
b15
CarryIn
ALU3
P3
G3
Result12–15
pi + 3
gi + 3
C4
ci + 4
CarryOut

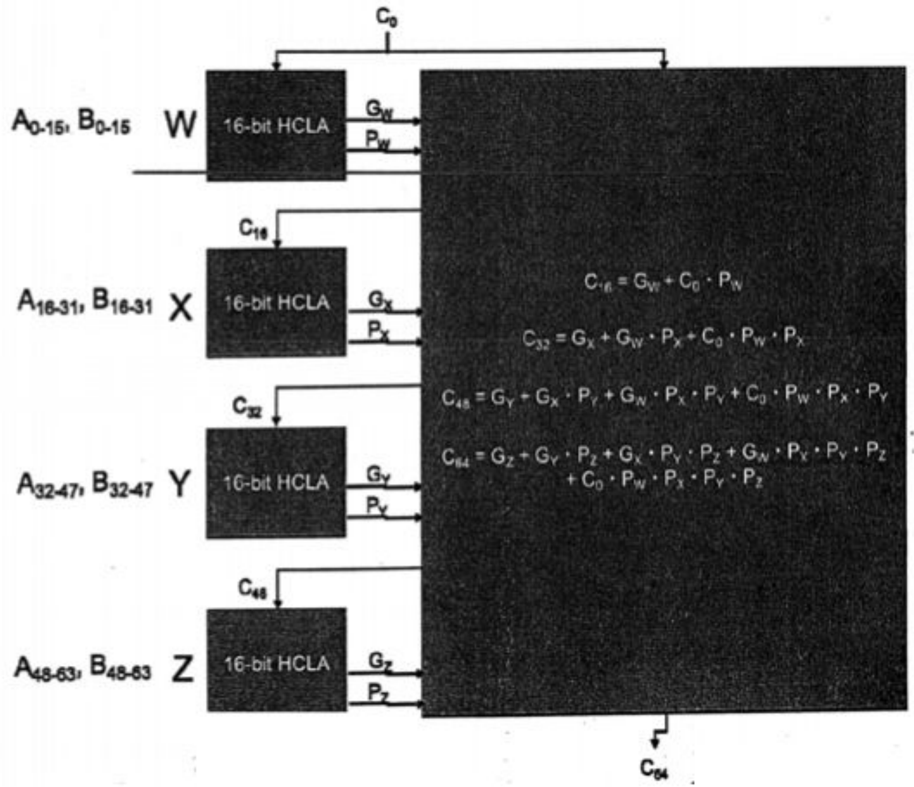
P1 - HCLA

2) Find the delay for calculating the following signals:

G0	
P0	
G3	
P3	
G_δ	
P_δ	

P_W	
G_W	
P_Z	
G_Z	
C48	
C64	
C60	
C63	
S63	

fan-in	AND/ OR	XOR
2	1T	2T
3	2T	3T
4	5T	6T
5	7T	9T
6	10T	12T
7	13T	16T
8	15T	18T



C_0
$A_{0\text{-}15}, B_{0\text{-}15}$ W
16-bit HCLA
G_W
P_W
C_{16}
$A_{16\text{-}31}, B_{16\text{-}31}$ X
16-bit HCLA
G_X
P_X
C_{32}
$A_{32\text{-}47}, B_{32\text{-}47}$ Y
16-bit HCLA
G_Y
P_Y
C_{48}
$A_{48\text{-}63}, B_{48\text{-}63}$ Z
16-bit HCLA
G_Z
P_Z
$C_{16} = G_W + C_0 \cdot P_W$
$C_{32} = G_X + G_W \cdot P_X + C_0 \cdot P_W \cdot P_X$
$C_{48} = G_Y + G_X \cdot P_Y + G_W \cdot P_X \cdot P_Y + C_0 \cdot P_W \cdot P_X \cdot P_Y$
$C_{64} = G_Z + G_Y \cdot P_Z + G_X \cdot P_Y \cdot P_Z + G_W \cdot P_X \cdot P_Y \cdot P_Z + C_0 \cdot P_W \cdot P_X \cdot P_Y \cdot P_Z$
C_{64}

This question was collected from a previous midterm exam

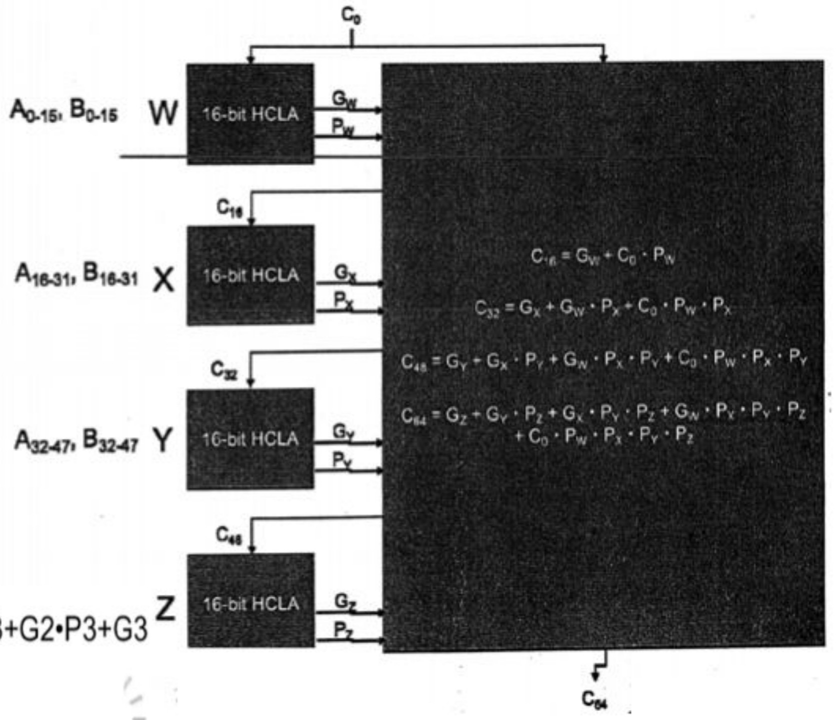
P1 - HCLA

2) Find the delay for calculating the following signals:

G0	1T
P0	2T
G3	1T
P3	2T
G_δ	12T
P_δ	7T

P_W	12T
G_W	22T
P_Z	12T
G_Z	22T
C48	29T
C64	34T
C60	39T
C63	49T
S63	51T

fan-in	AND/ OR	XOR
2	1T	2T
3	2T	3T
4	5T	6T
5	7T	9T
6	10T	12T
7	13T	16T
8	15T	18T


C_0
$A_{0\text{-}15}, B_{0\text{-}15}$ W
16-bit HCLA
G_W
P_W
C_{16}
$A_{16\text{-}31}, B_{16\text{-}31}$ X
16-bit HCLA
G_X
P_X
C_{32}
$A_{32\text{-}47}, B_{32\text{-}47}$ Y
16-bit HCLA
G_Y
P_Y
C_{48}
Z
16-bit HCLA
G_Z
P_Z
$C_{16} = G_W + C_0 \cdot P_W$
$C_{32} = G_X + G_W \cdot P_X + C_0 \cdot P_W \cdot P_X$
$C_{48} = G_Y + G_X \cdot P_Y + G_W \cdot P_X \cdot P_Y + C_0 \cdot P_W \cdot P_X \cdot P_Y$
$C_{64} = G_Z + G_Y \cdot P_Z + G_X \cdot P_Y \cdot P_Z + G_W \cdot P_X \cdot P_Y \cdot P_Z + C_0 \cdot P_W \cdot P_X \cdot P_Y \cdot P_Z$
C_{64}

- $G = G0 \cdot P1 \cdot P2 \cdot P3 + G1 \cdot P2 \cdot P3 + G2 \cdot P3 + G3$
- $P = P0 \cdot P1 \cdot P2 \cdot P3$

Please watch discussion video for detailed proof! (Student's proofs are correct)

This question was collected from a previous midterm exam

P2 - Multiplication

Calculate `1001 * 1101` by left-shifting the multiplicand. (as unsigned)

Repeat, but right-shift the product this time.

P3 - Multiplication

Calculate `1001 * 1101` (as signed using booth's algorithm)

P4 (review, if time permits)

Assume a program requires the execution of 50×10^6 FP instructions, 110×10^6 INT instructions, 80×10^6 L/S instructions, and 16×10^6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1) By how much must we improve the CPI of FP instructions if we want the program to run two times faster?
2) By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

P4 (review, if time permits)

Assume a program requires the execution of 50×10^6 FP instructions, 110×10^6 INT instructions, 80×10^6 L/S instructions, and 16×10^6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1) By how much must we improve the CPI of FP instructions if we want the program to run two times faster? ans: It is not possible. The total cycles needed for FP instructions is less than half of total cycles required for the whole program so even if we made FP have a CPI of 0, the program could not be run in less than half the time.
2) By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

CPI for FP: 0.6
CPI for INT: 0.6
CPI L/S : 4 * 0.7 = 2.8
CPI for branch: 1.4

Original T = # instructions * # cycles per instruction * seconds per cycle = (50E6 * 1 + 110E6 * 1 + 80E6 * 4 + 16E6 * 2) / 2E9 = 0.256 s

New T = # instructions * # cycles per instruction * seconds per cycle = (50E6 * 0.6 + 110E6 * 0.6 + 80E6 * 2.8 + 16E6 * 1.4) / 2E9 = 0.171 s

Original T / New T ~ 1.50 times speedup.

P4 (review, if time permits)

If the current value of the PC is 0x0000 0008, can you use a single jump instruction to get to the PC addresses below?

1) 0010 0000 0000 0001 0100 1001 0010 0100 ?

   No, we can't, as the upper four bits of our current PC is 0000, but the target's upper four bits are 0010.

2) 0x0000 0600 ?

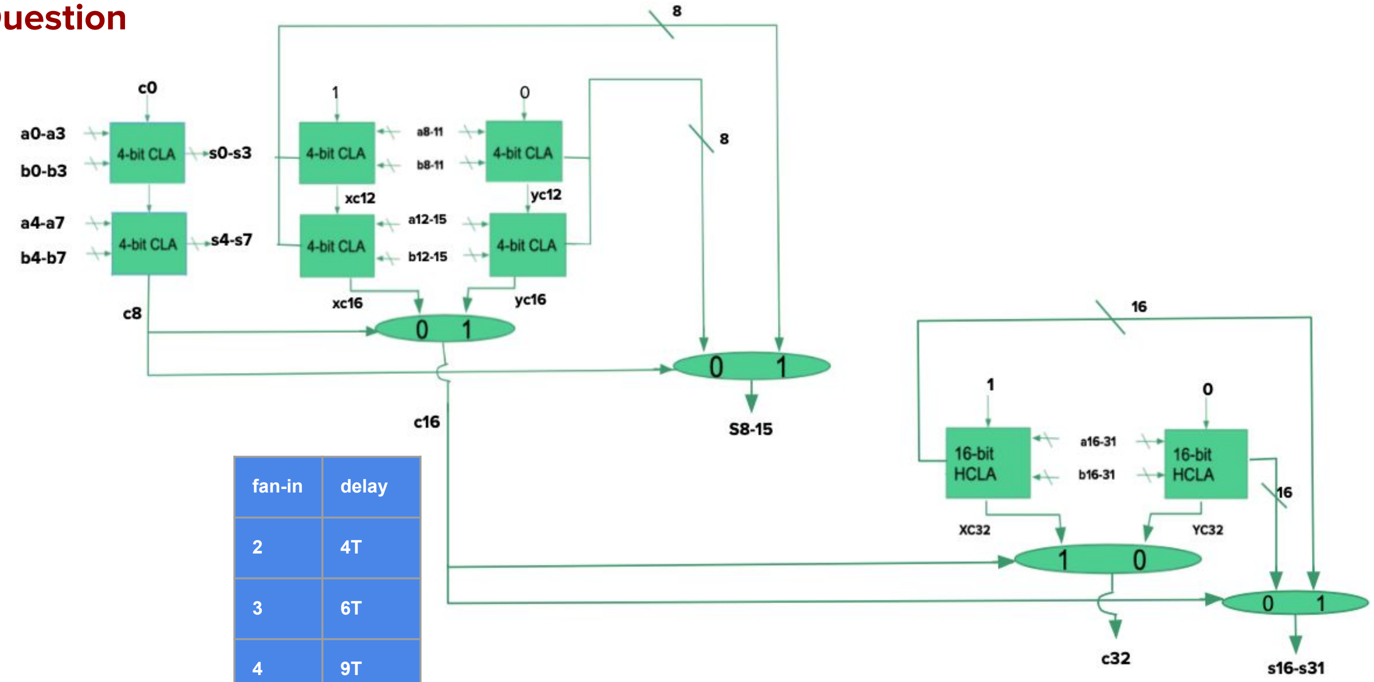
   Yes, this is possible. The most significant four bits of the target address is: 0000, which is the same as our PC's upper four bits, the lowest two bits of the target is 00, thus we can jump to this address in a single jump.

Optional Question

G0	4T
P0	4T
G3	4T
P3	4T
C3	22T
S3	26T
C4	30T
G8	4T
P8	4T

G12	4T
P12	4T
C8	56T
C16 (after mux)	68T
S15(after mux)	68T

G_8	22T
P_8	13T
C28	37T
C31	55T
C32(before)	44T
C32(after)	80T
S31(before)	59T
S31(after)	80T
Maximal delay	80T



fan-in	delay
2	4T
3	6T
4	9T
5	13T
6	17T
7	22T
8	28T

Mux has delay 12T

This question was collected from a previous midterm exam

Q&A

Any questions ? Feedback?

Thanks!

Acknowledgement

Patterson, David, and John Hennessy. *Computer Organization and Design MIPS Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. 5th ed., Morgan Kaufmann, 2013.

Credit to Prof. (Glenn) Reinman (some practice questions were extracted from previous midterm exams)

Booth’s algorithm - example(1)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
5*3 = 15

    0101
    0011(0)
----------------
   -0101              1111 1011
   0000                  0 000
  +0101        =        01 01
  0000                 000 0
                     ----------------
                      0000 1111

                      =15
```

Booth's algorithm - example(2)

```
key:
00 = shift
11 = shift
10 = subtract
01 = add
```

```
7*(-3) = -21

    0111
    1101(0)
-----------------
   -0111
  +0111
 -0111
 0000
```

=

```
1111 1011
   0 000
  01 01
 000 0
-----------------
0000 1111
```

=15