Lecture 8: Computer Arithmetic

- Last Time
  - Exceptions
  - Compiler's use of the ISA
  - Start of computer arithmetic

- Today
  - RISC vs. CISC + discussion
  - Carry-lookahead addition, shift, multiplication
  - Datapath organization

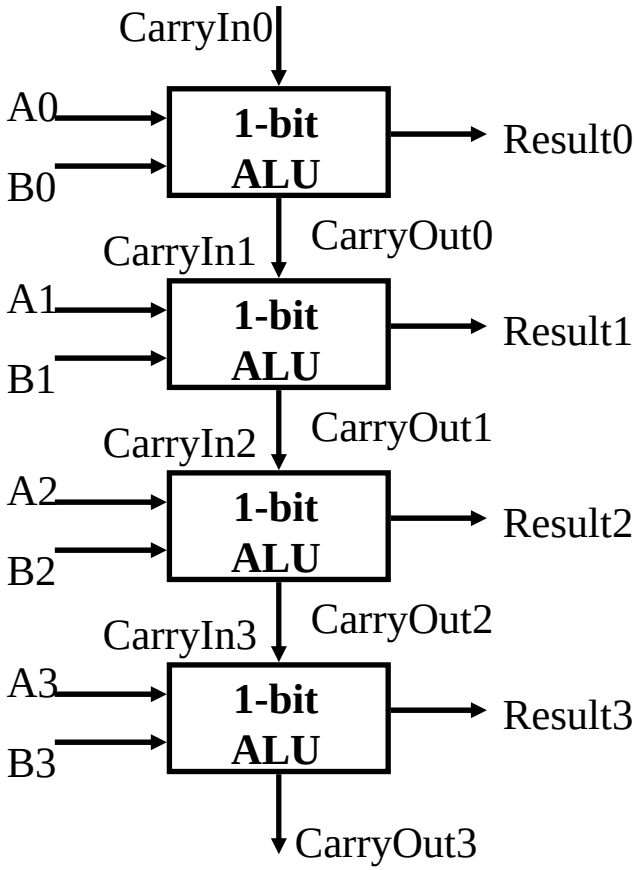
CISC vs RISC

- What is a RISC?
  (Reduced Instruction Set Computer)
  - no firm definition
  - generally includes
    - general registers
    - fixed 3-address instruction format
    - strict load-store architecture
    - simple addressing modes
    - simple instructions
  - Examples
    - DEC Alpha
    - MIPS
  - Advantages
    - good compiler target
    - easy to implement/pipeline

- CISC (Complex Instruction-Set Computer)
  - CISC ≡ ¬RISC
  - may include
    - variable length instructions
    - memory-register instructions
    - complex addressing modes
    - complex instructions
      - CALLP, EDIT, …
  - Examples
    - DEC VAX, IBM 370, x86
  - Advantages
    - better code density
    - legacy software

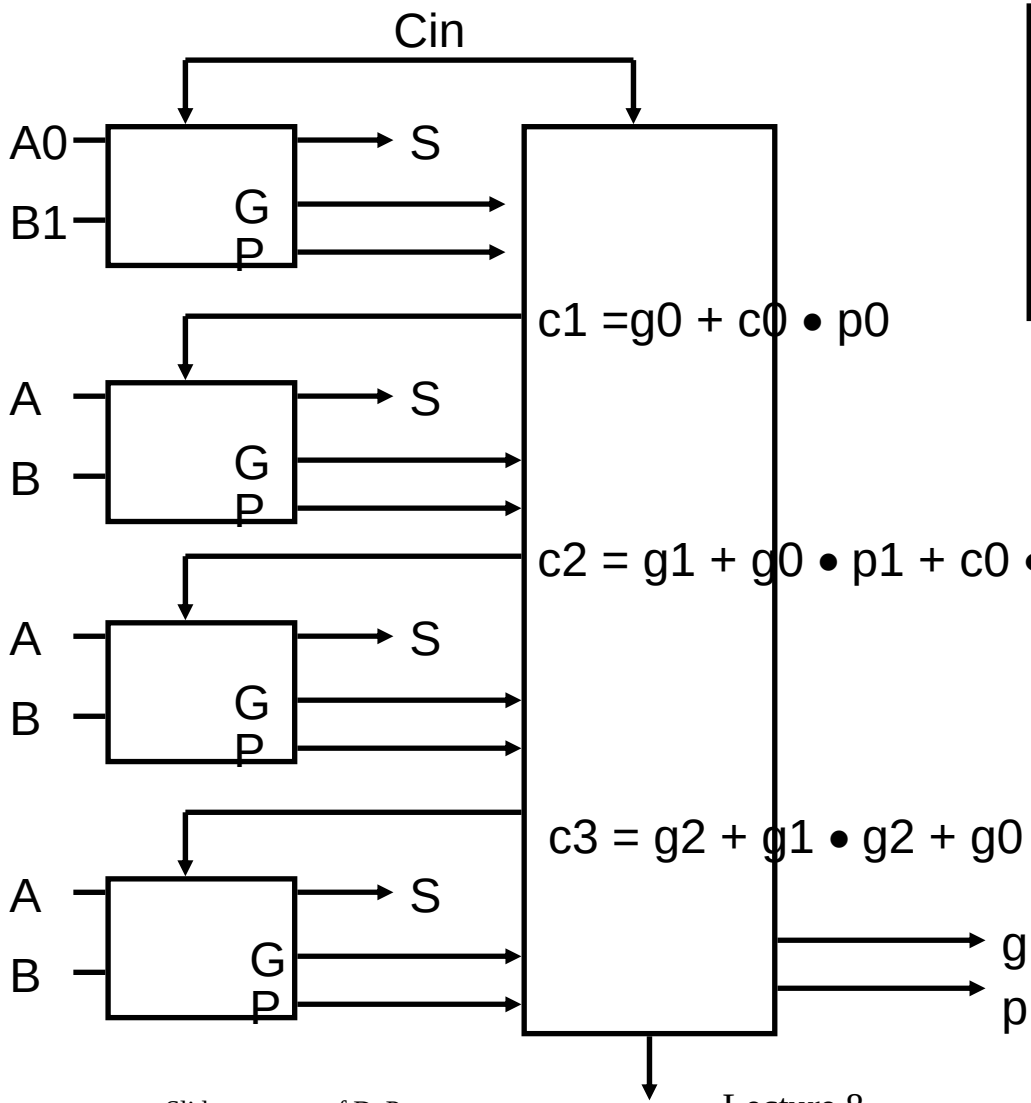
But What about Performance?

- Critical Path of n-bit Rippled-carry adder is n*CP



Design Trick: throw hardware at it

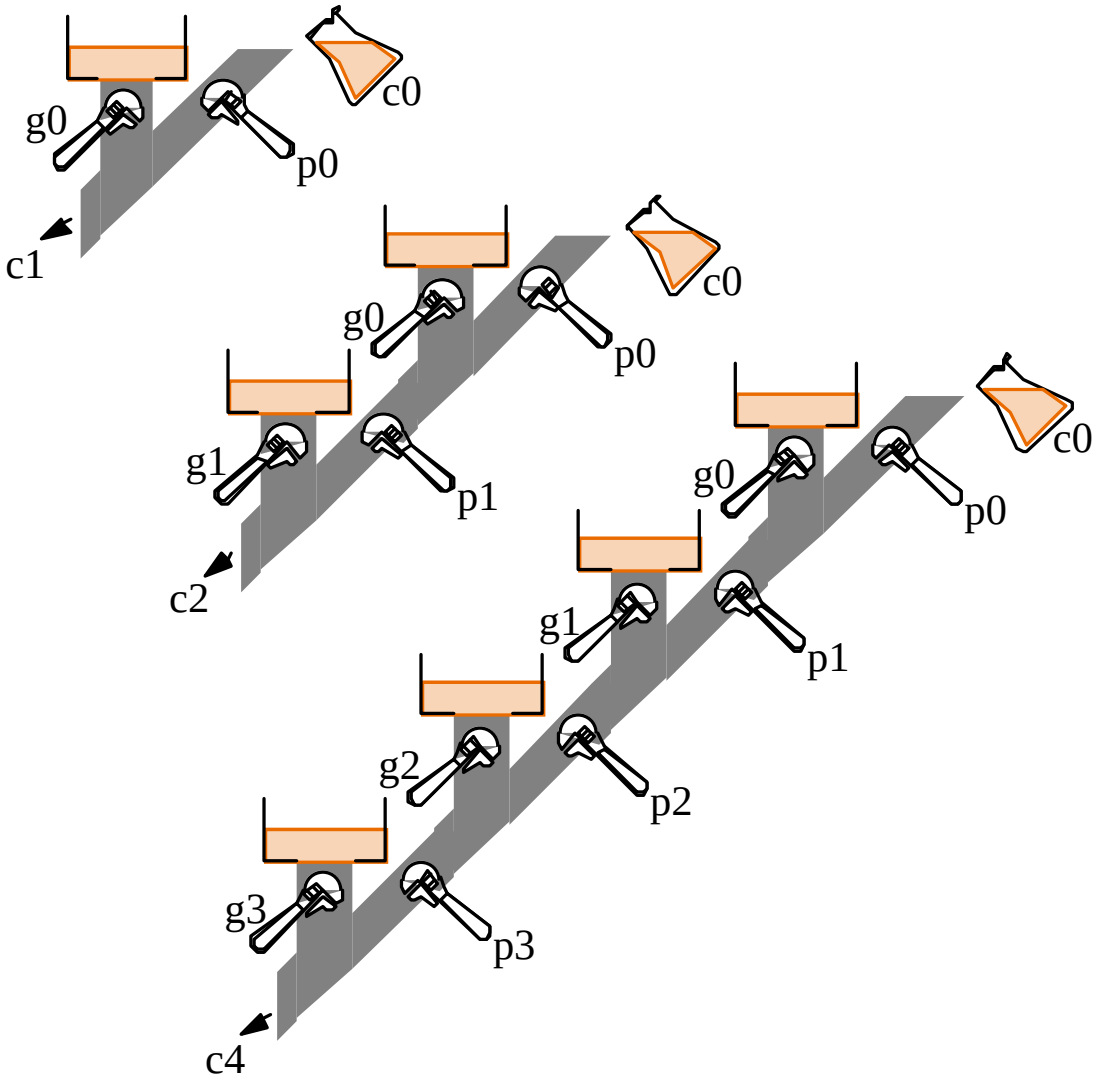
Carry Look Ahead (Design trick: peek)



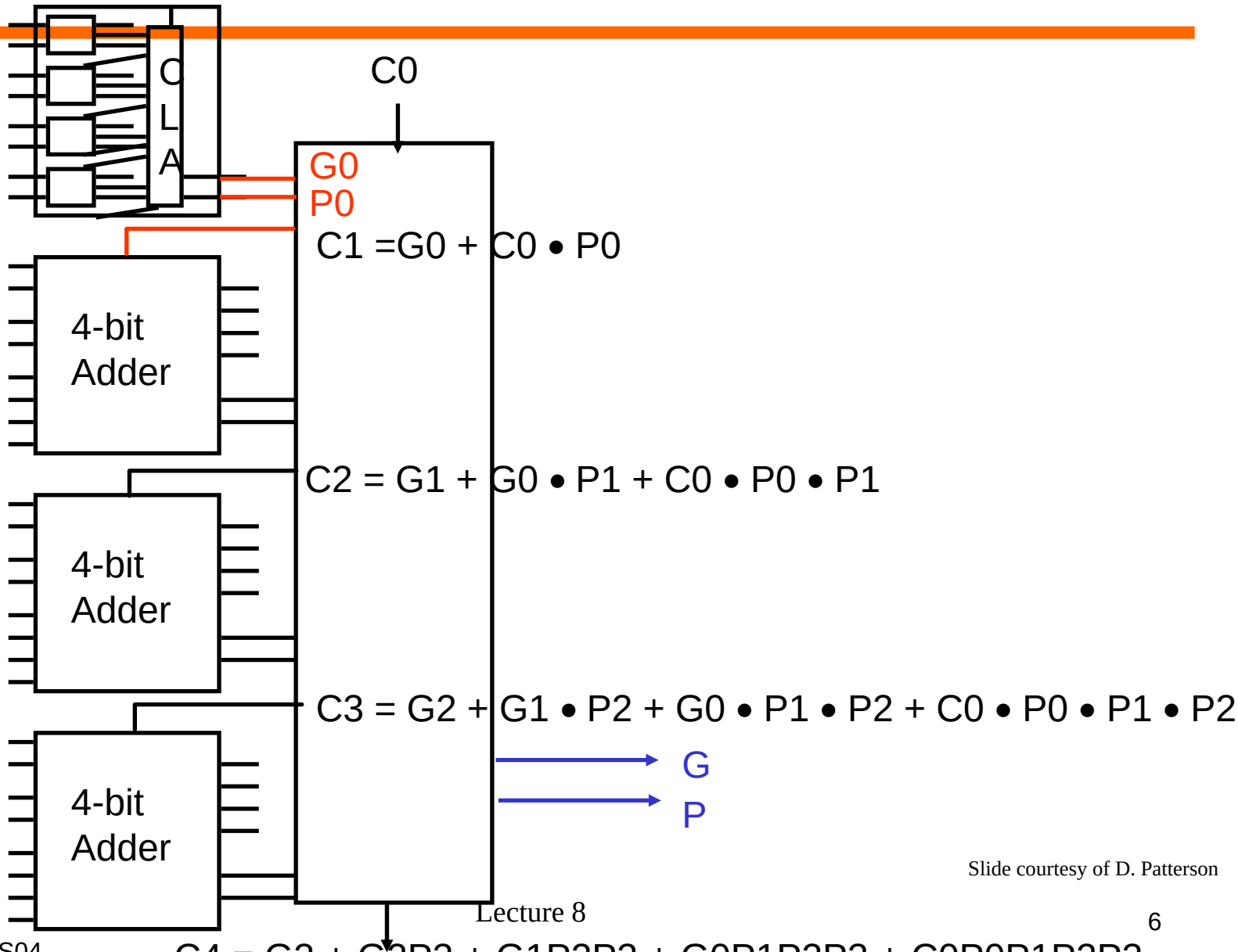
A	B	C-out	
0	0	0	"kill"
0	1	C-in	"propagate"
1	0	C-in	"propagate"
1	1	1	"generate"

P = A and B
G = A xor B

Plumbing as Carry Lookahead Analogy

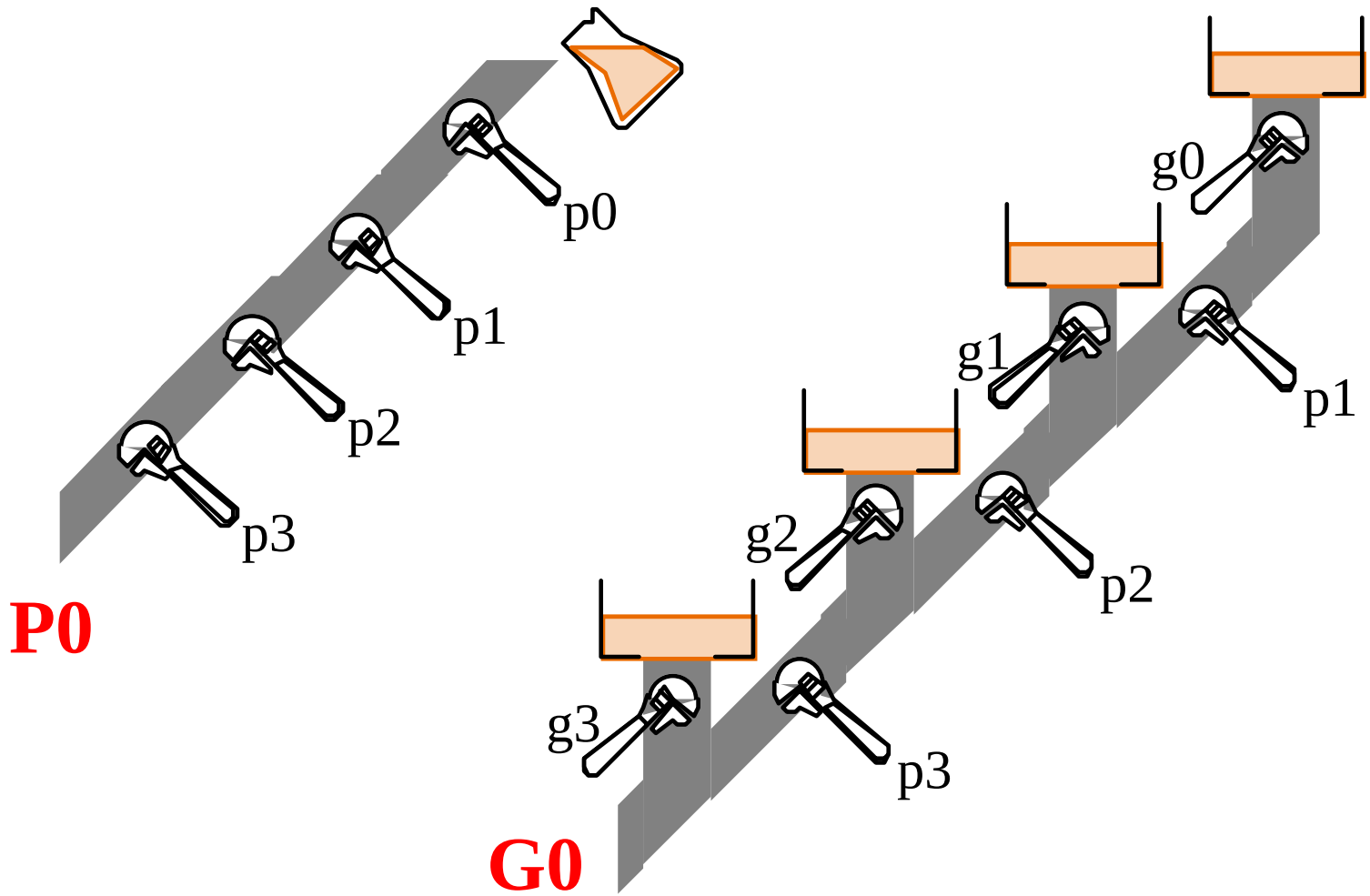


Cascaded Carry Look-ahead (16-bit): Abstraction



Slide courtesy of D. Patterson

2nd level Carry, Propagate as Plumbing

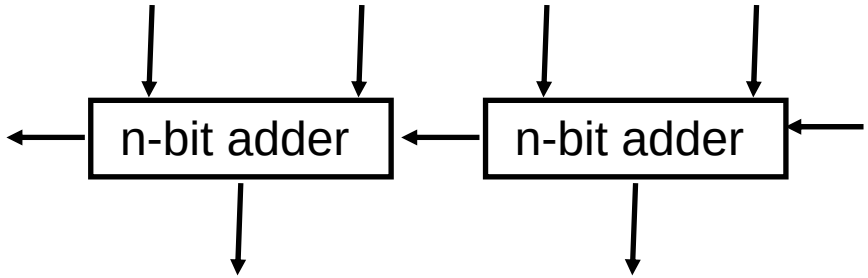


Delay Analysis

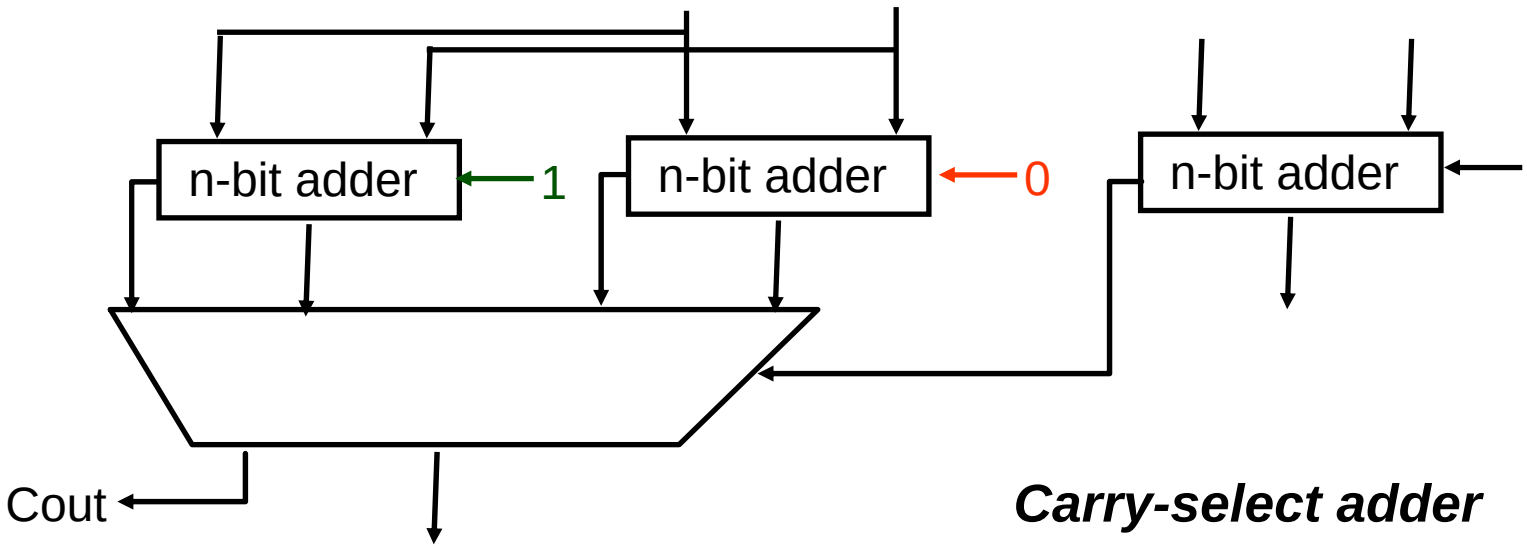
- 16-bit ripple carry adder
  - $T = 16 * T_{FA}$
    $= 16 * 2$ gate delays $= 32$ gate delays
- 16-bit carry lookahead adder
  - $T = T_{C4}$
    $= 2 + \max(P_i, G_i) = 2 + T_{G0}$
    $= 2 + 2 + \max(p_i, g_i)$
    $= 5$ gate delays
  - Real designs use some nice transistor circuits for the carry lookahead chain
- In the limit – expand to full lookahead
  - T = 2 – but: a lot of logic, very wide fan-in gates
  - too costly

Design Trick: Guess

T(2n) = 2*T(n)



T(2n) = T(n) + T(mux)



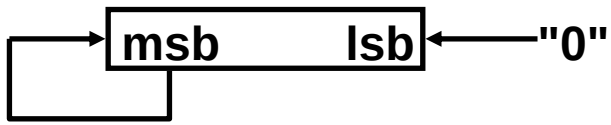
Carry-select adder

Shifters

Two kinds:

***logical*-- value shifted in is always "0"**

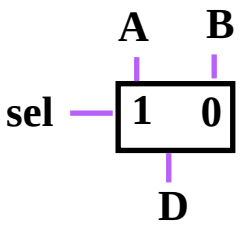
***arithmetic*-- on right shifts, sign extend**



Note: these are single bit shifts. A given instruction might request 0 to 32 bits to be shifted!

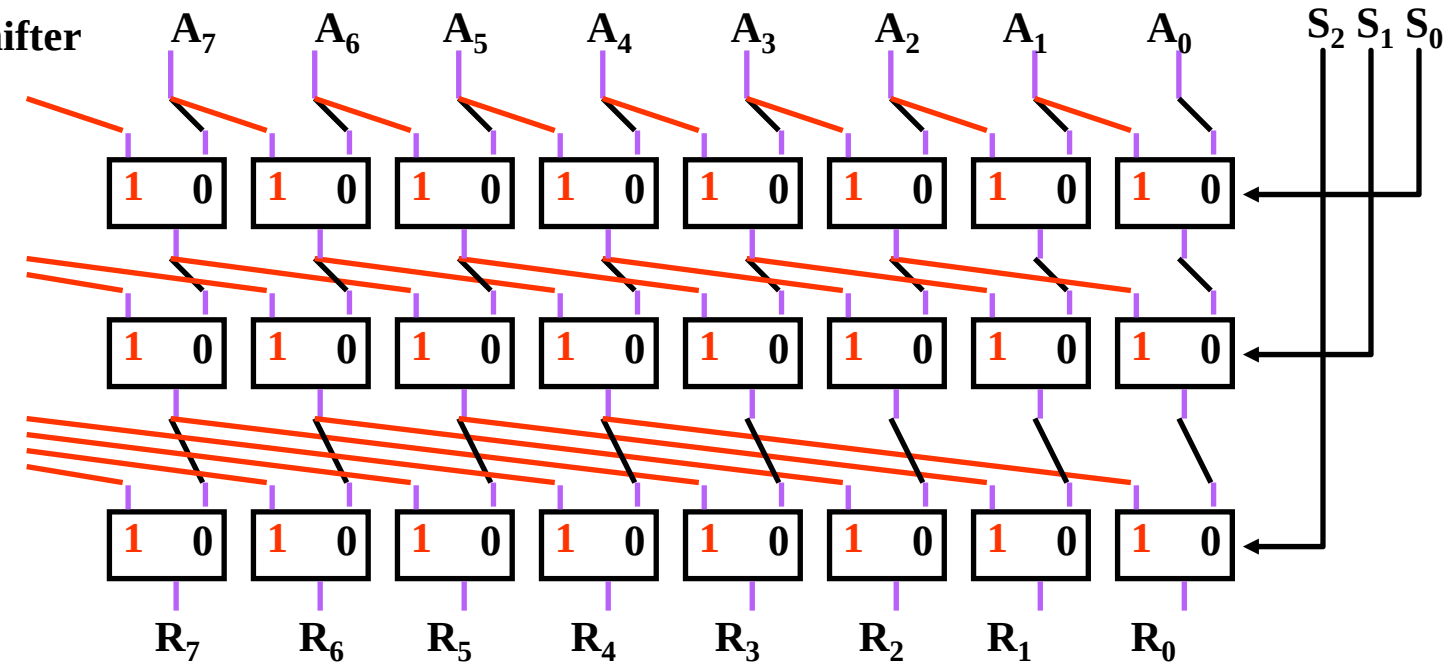
Combinational Shifter from MUXes

Basic Building Block

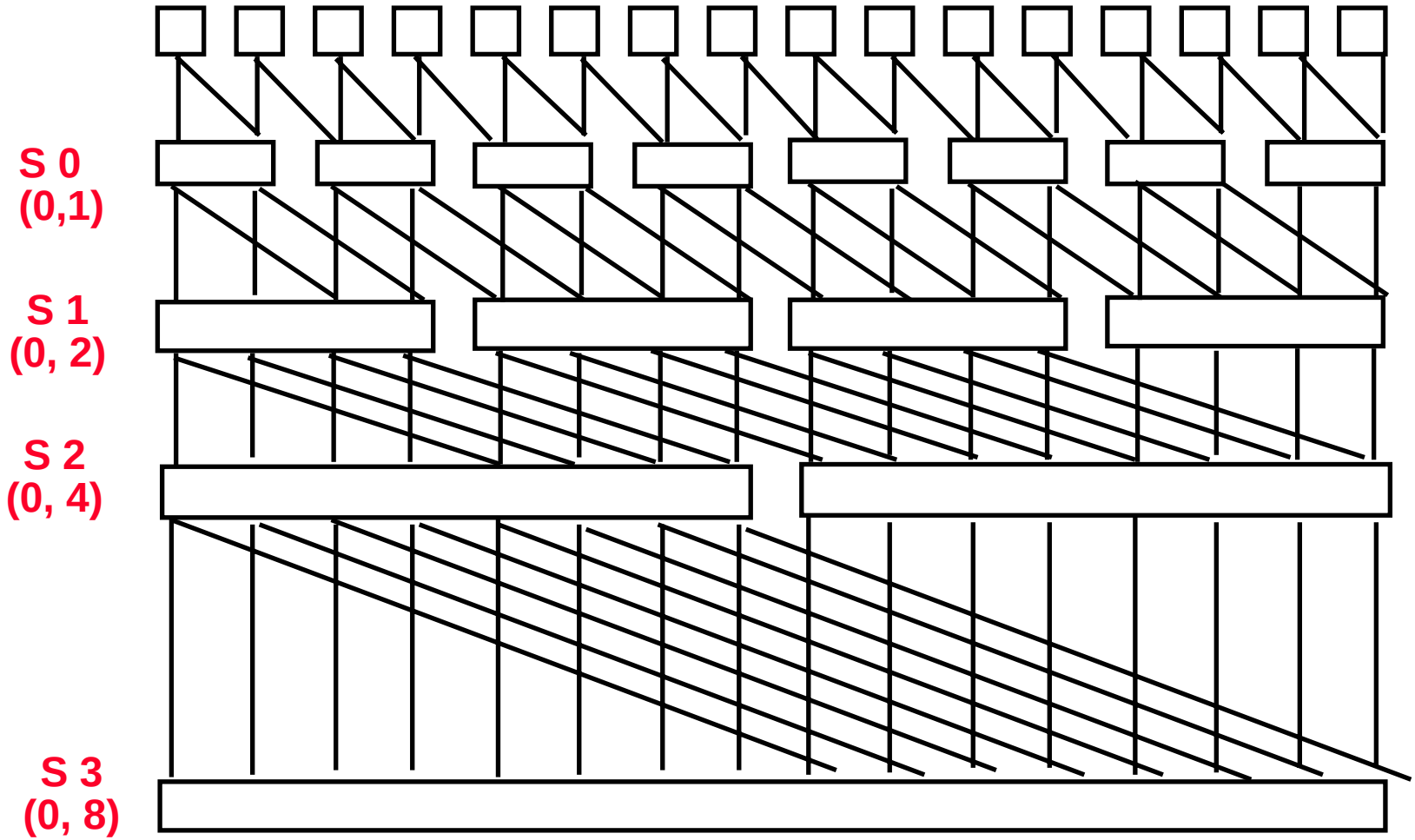


- What comes in the MSBs?
- How many levels for 32-bit shifter?
- What if we use 4-1 Muxes ?

8-bit right shifter

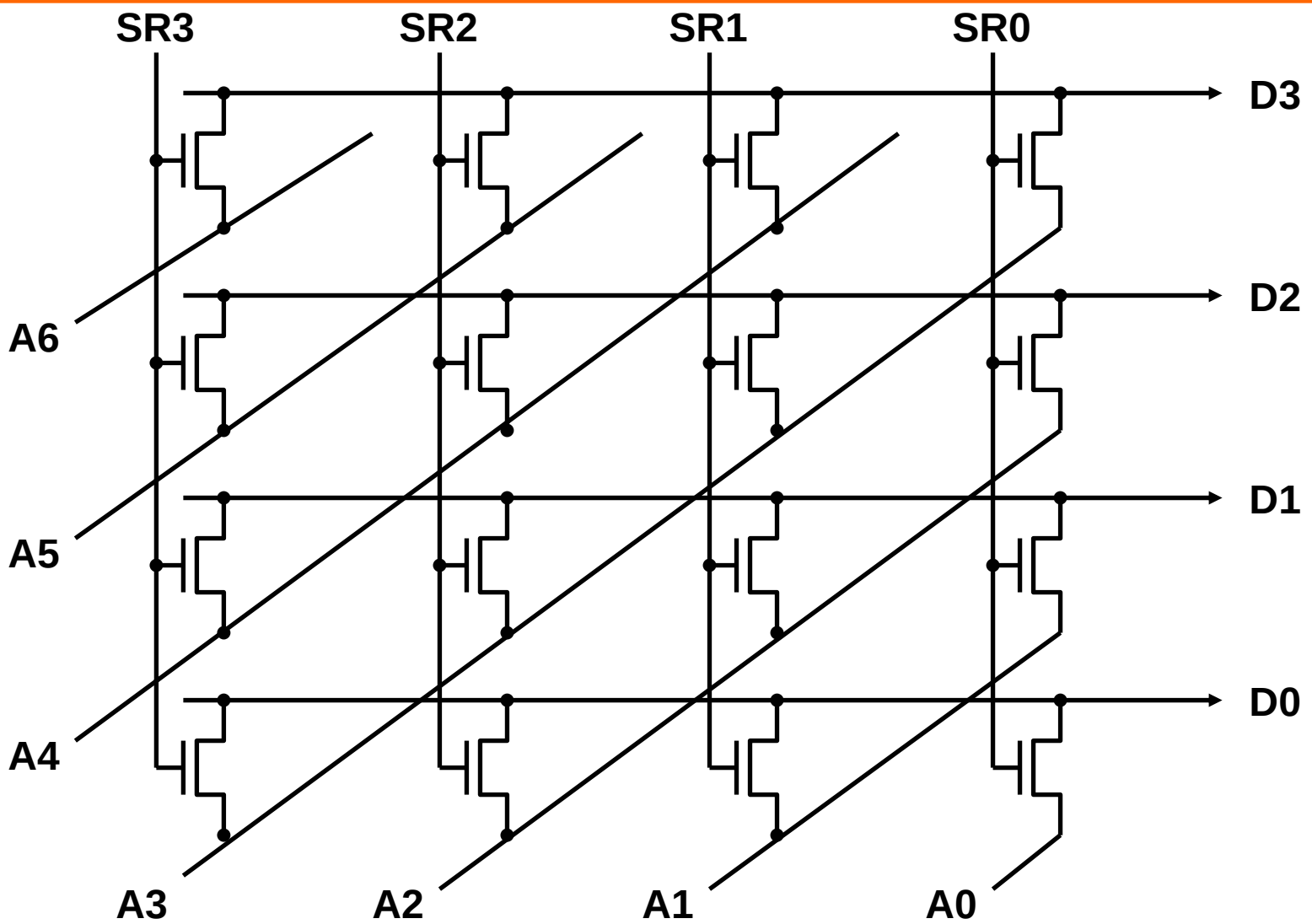


General Shift Right Scheme using 16 bit example



If added Right-to-left connections could support Rotate (not in MIPS but found in ISAs)

Barrel Shifter – One transistor per switch



MULTIPLY (unsigned)

- Paper and pencil example (unsigned):

Multiplicand 1000

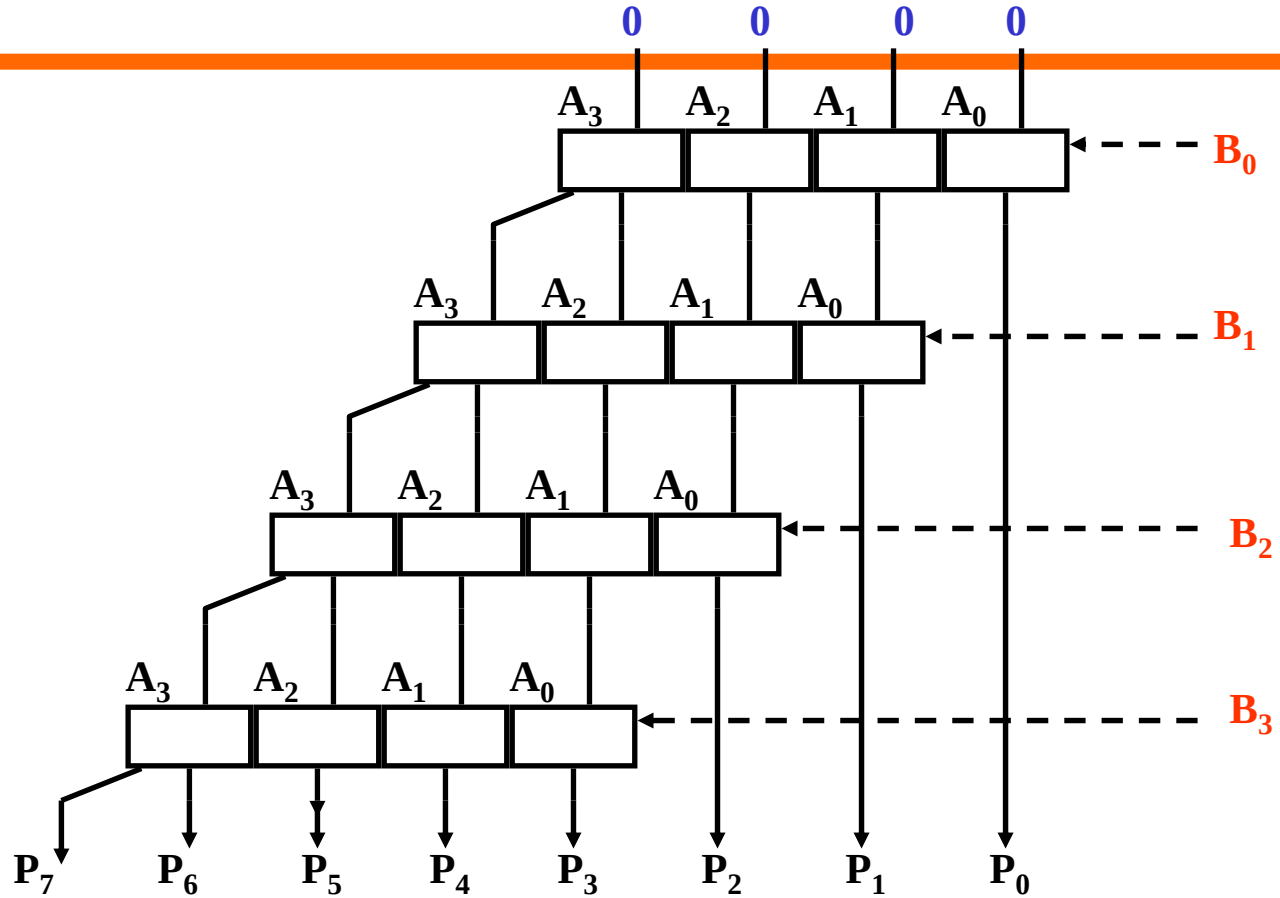
Multiplier 1001

```
  1000
  1000
 0000
0000
1000
01001000
```

Product

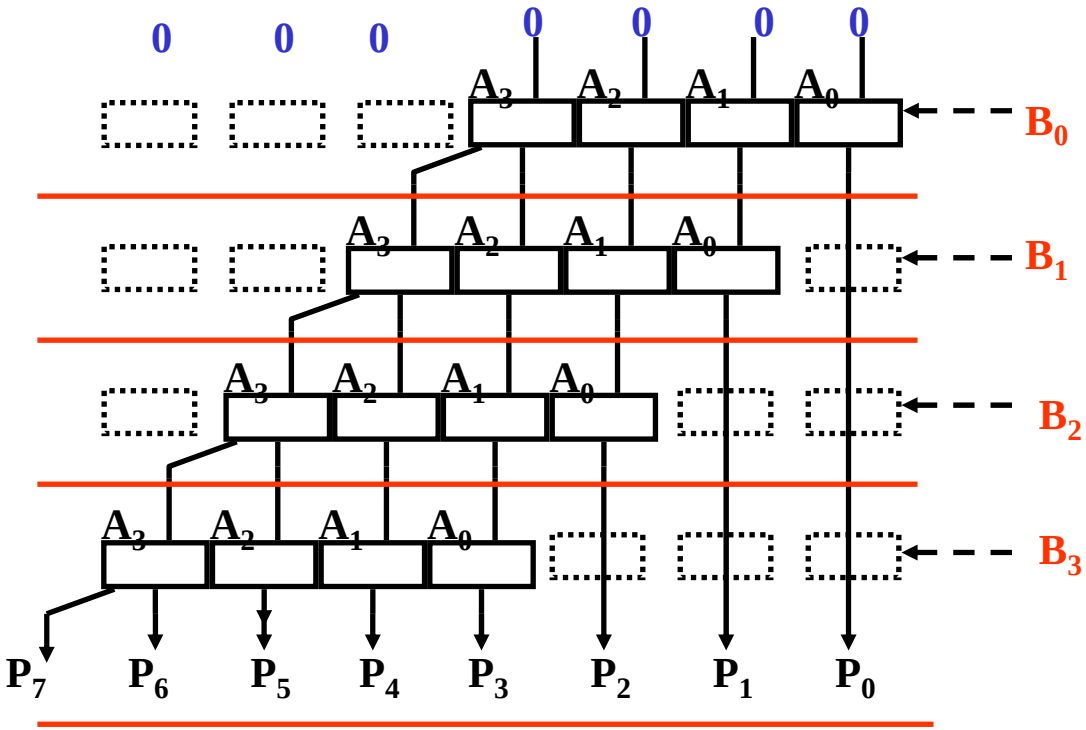
- m bits x n bits = m+n bit product
- Binary makes it easy:
  - 0 => place 0 (0 x multiplicand)
  - 1 => place a copy (1 x multiplicand)
- 4 versions of multiply hardware & algorithm:
  - successive refinement

Unsigned Combinational Multiplier



- Stage i accumulates $A * 2^i$ if $B_i == 1$
- Q: How much hardware for 32 bit multiplier? Critical path?

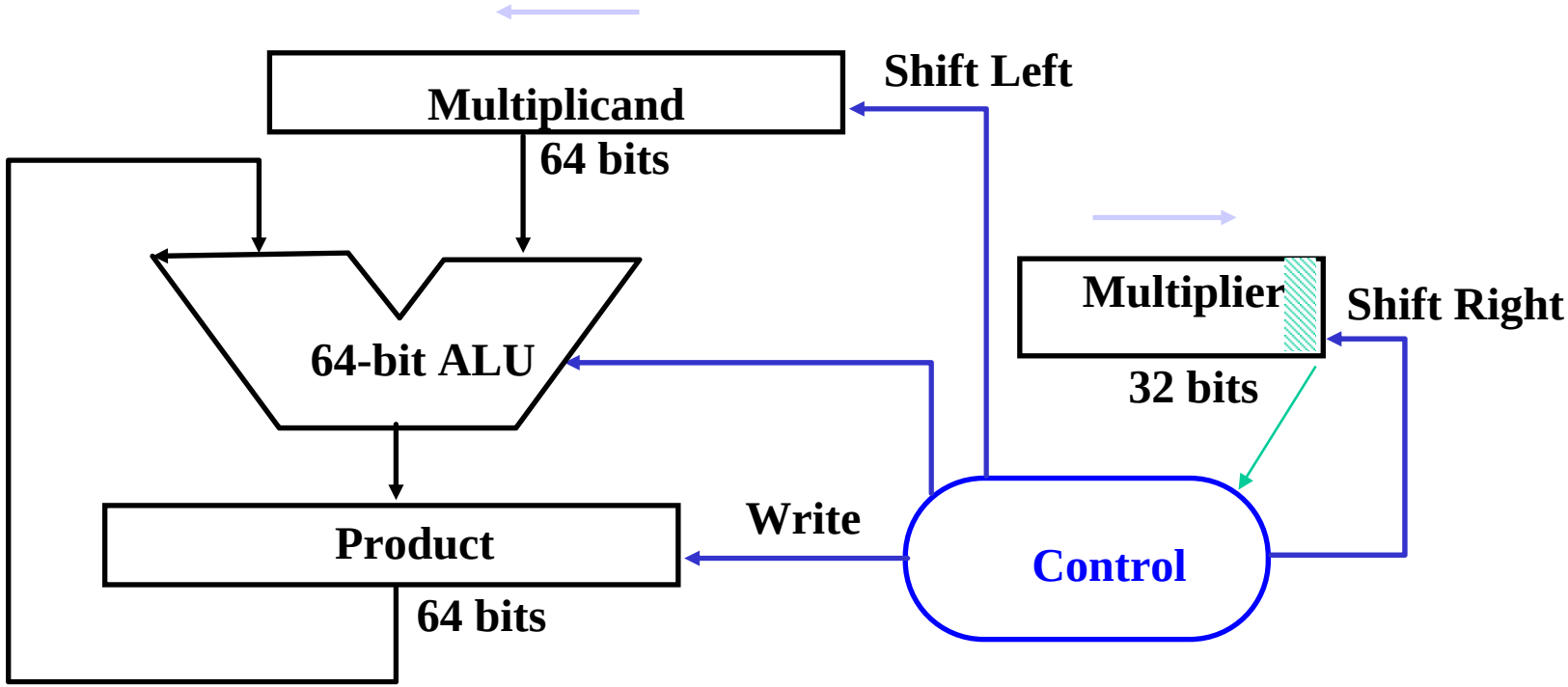
How does it work?



- at each stage shift A left (x 2)
- use next bit of B to determine whether to add in shifted multiplicand
- accumulate 2n bit partial product at each stage

Unisigned shift-add multiplier (version 1)

- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit multiplier reg



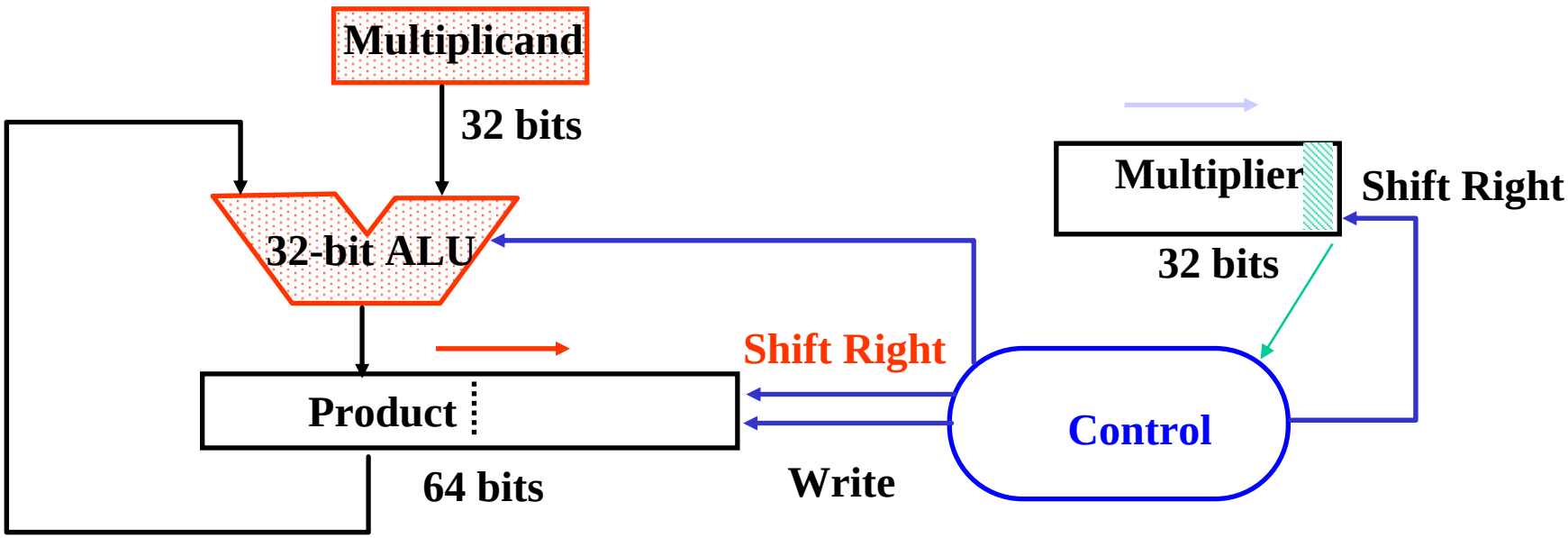
Multiplier = datapath + control

Observations on Multiply Version 1

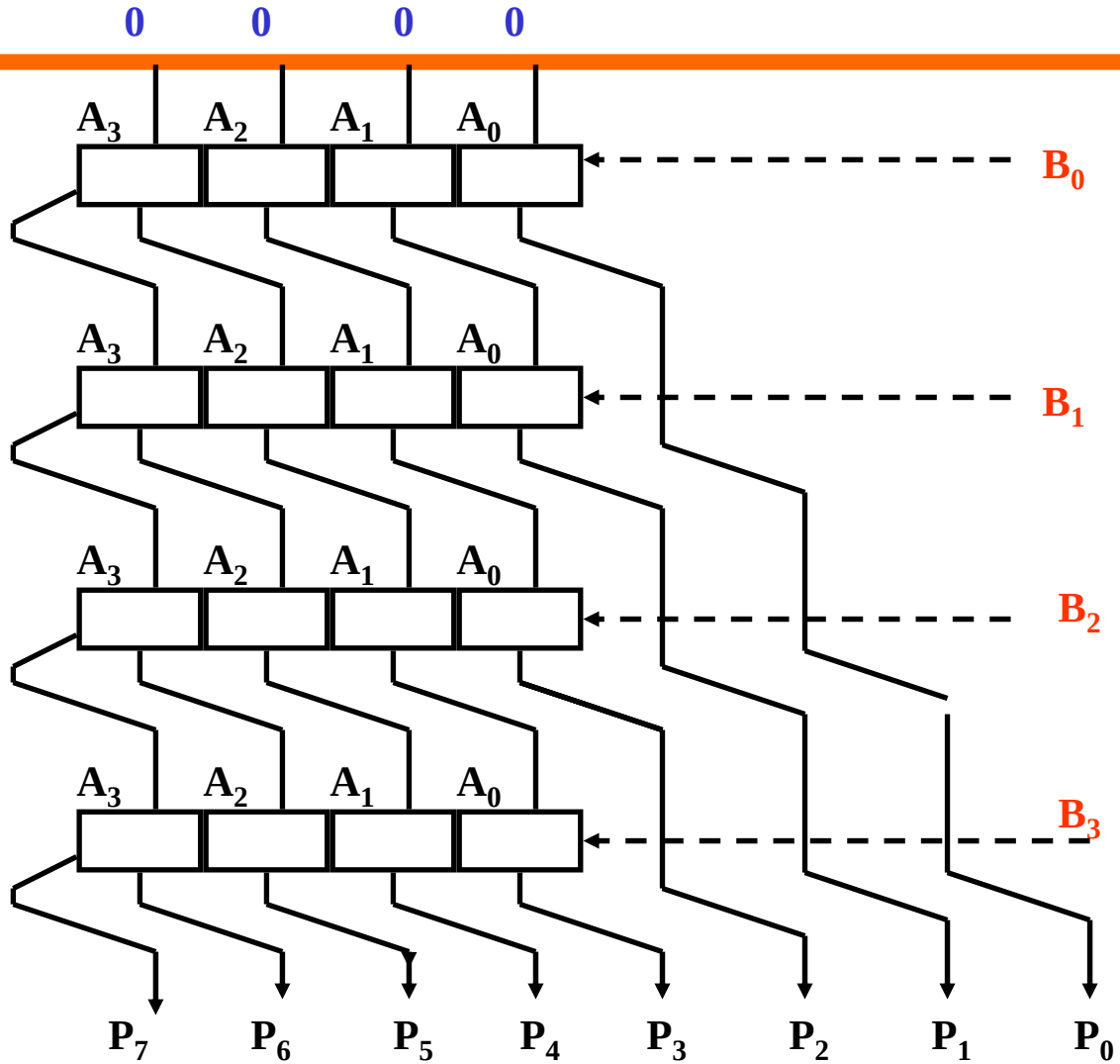
- 1 clock per cycle => - 100 clocks per multiply
  - Ratio of multiply to add 5:1 to 100:1
- 1/2 bits in multiplicand always 0
  => 64-bit adder is wasted
- 0's inserted in left of multiplicand as shifted
  => least significant bits of product never changed once formed
- Instead of shifting multiplicand to left, shift product to right?

MULTIPLY HARDWARE Version 2

- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, 32-bit Multiplier reg



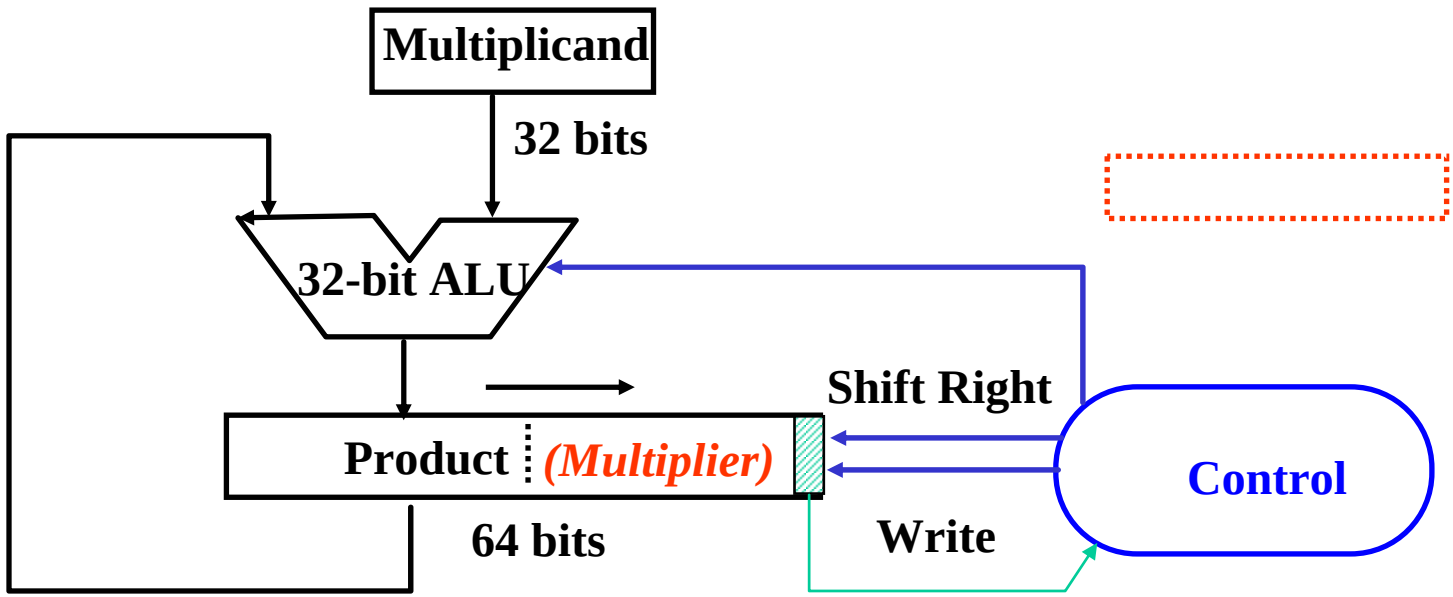
What's going on?



- Multiplicand stay's still and product moves right

MULTIPLY HARDWARE Version 3

- Still wasting some storage space
  - Bits of multiplier used one-by-one, can overlap with product
- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, (0-bit Multiplier reg)



Observations on Multiplication

- Can speed up algorithm by doing 2 bits at a time, instead of just one
  - How????
  - See Booth encoding strategy in chapter 4
- Multiplication algorithm is still slow
  - Each step is doing a full carry-propagate add
  - Can use carry save adders instead
  - Build a Wallace tree to combine the partial products
  - Single carry-propagate add instead
- Division algorithm is very similar to multiplication
  - See section 4.7

Summary

- Add/Shift/Multiply

- Next Time
  - Pipelining basics
  - Data/Control Hazards - up close and personal
  - Pipelining Multicycle instructions

- Reading assignment – P&H 4.8-4.12 (skim)
  P&H Chapter 5 (skim)