

ALU Design

Lecturer: Dr. Hui Annie Guo
huig@cse.unsw.edu.au
K17-501F (ext. 57136)

Overview

- Design process
- Design approaches
- Typical design techniques/tricks
 - ALU
 - Integer Multiplier
 - Shifter

COMP3211/9211

2011S1 wk3_1 P2

Design Process

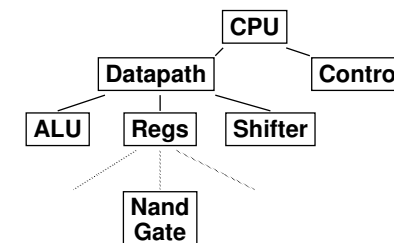
- Design begins with requirements
 - Functional capabilities: what it will do
 - Performance characteristics: speed, power, area, . . .
- Design finishes as assembly
 - A representation that describes components used and how those components are assembled in order to meet design requirements
- Design is a "creative process," not a simple, linear process
 - May need to iterate many times to achieve a successful design

COMP3211/9211

2011S1 wk3_1 P3

Design Approaches

- Two basic approaches used in design:
 - Top-Down decomposition of complex functions (behaviors) into more primitive functions [analysis]
 - Bottom-Up composition of primitive building blocks into more complex assemblies [synthesis]

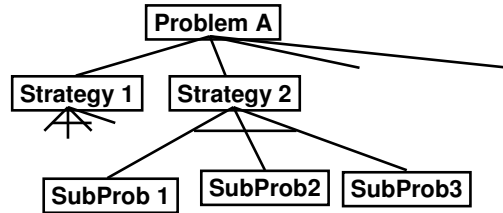


- The real design approach used by designers may be a creative employment of both Top-Down and Bottom-Up approaches in the design process

COMP3211/9211

2011S1 wk3_1 P4

Design as Search



- **Design involves educated guesses and verification**
 - Given the goals, how should these be prioritized?
 - Given alternative design pieces, which should be selected?
 - Given design space of components & assemblies, which part will yield the best solution?

Feasible (good) choices vs. Optimal choices

Problem: Design a “fast” ALU for the MIPS ISA

- **Requirements**
 - Must support the Arithmetic / Logic operations
 - Tradeoff of cost and speed based on frequency of occurrence, hardware budget

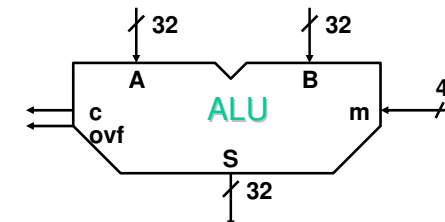
MIPS ALU requirements

- **Add, AddU, Sub, SubU, AddI, AddIU**
 - 2’s complement adder/sub with overflow detection
- **And, Or, AndI, OrI, Xor, Xori**
 - Logical AND, logical OR, XOR
- **SLT, SLTI, SLTIU (set less than)**
 - 2’s complement adder with inverter, check sign bit of result
- **Mult, MultU, Div, DivU**
 - 32-bit multiply and divide, signed and unsigned
- **Sll, Srl, Sra**
 - left shift, right shift, right shift arithmetic by 0 to 31 bits

Please refer to “MIPS reference data” and Ch. 2 of P&H textbook for the instruction definition

Abstraction

- **Functional Specification**
 - inputs: 2 32-bit operands A, B, 4-bit mode
 - outputs: 32-bit result S, 1-bit carry, 1 bit overflow
 - operations: add, addu, sub, subu, and, or, xor, slt, sltu
- **Block Diagram (schematic symbol, VHDL entity)**



ALU Interface Description: VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Entity ALU is

    port ( signal A, B: in  std_logic_vector (0 to 31);
          signal m: in  std_logic_vector (0 to 3);
          signal S: out  std_logic_vector (0 to 31);
          signal c: out  std_logic;
          signal ovf: out  std_logic)
end ALU;
```

COMP3211/9211

2011S1 wk3_1 P9

Behavioral Representation: VHDL

```
Architecture fun_des of ALU is
Signal result: std_logic(32 downto 0);
Begin
    ...
    result <= ('0' & A) + ('0' & B);
    S <= result(31 downto 0);

    carry <= result(32);

    Ovf <= '1'
    when (A(31)=B(31))and (result(31)/= A(31))
    else '0';

    ...
End fun_des;
```

COMP3211/9211

2011S1 wk3_1 P10

How to design the ALU?

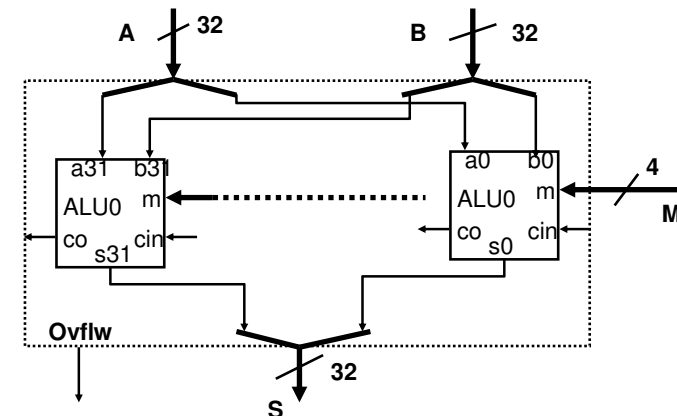
- **Divide and conquer**
 - Break the problem into simpler problems, solve them and glue together the solutions
 - E.g. divide a multiple bit operation into single bit operations.

COMP3211/9211

2011S1 wk3_1 P11

Bit Based Design

- **Design trick 1:** Design one “slice”, capable of computing the result for 2 1-bit inputs and duplicate the design 32 times

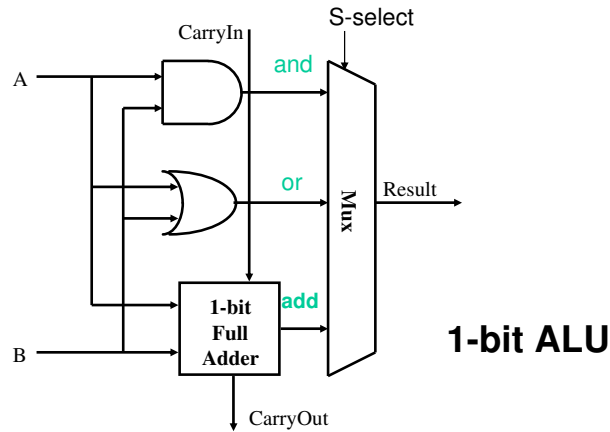


COMP3211/9211

2011S1 wk3_1 P12

Design with Building Blocks

- **Design trick 2:** take pieces you know (or can imagine) and try to put them together

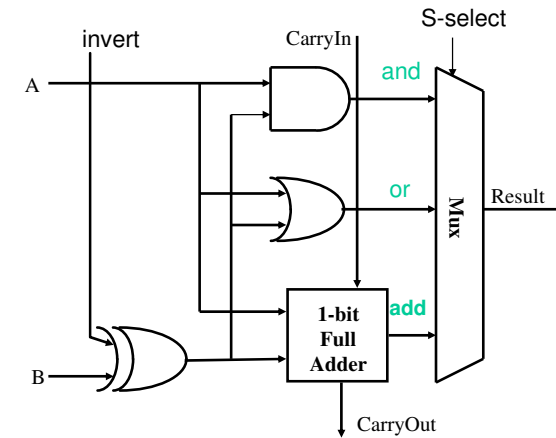


COMP3211/9211

2011S1 wk3_1 P13

Additional Operations (refinement)

- **Design trick 3:** solve part of the problem and extend
 - $A - B = A + (-B)$
 - form two's complement by invert and add one

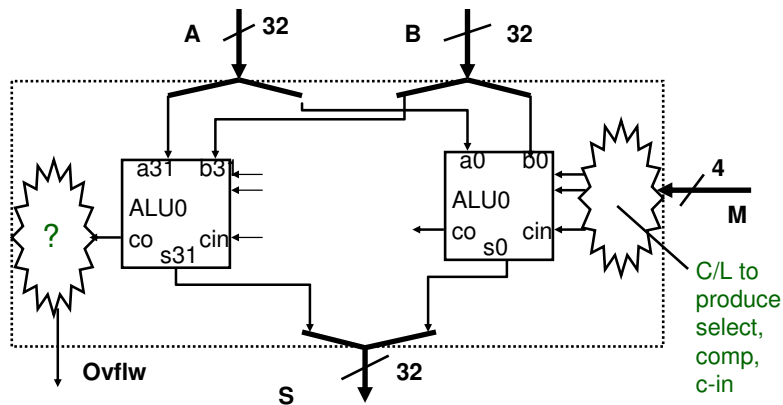


COMP3211/9211

2011S1 wk3_1 P14

Revised Diagram

- **LSB and MSB need to do a little extra**

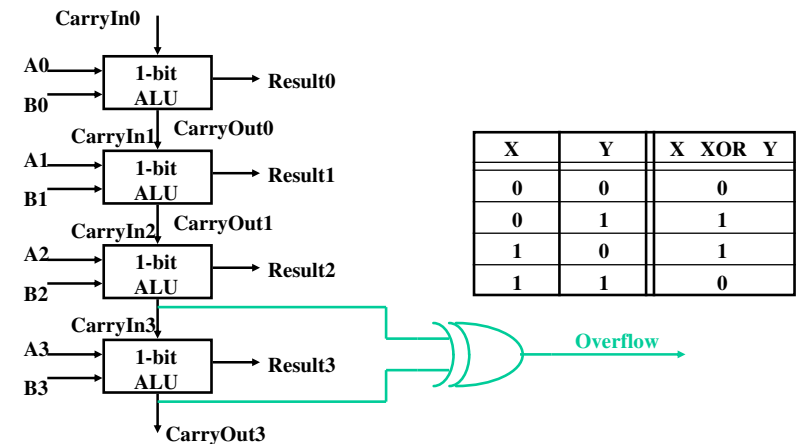


COMP3211/9211

2011S1 wk3_1 P15

Overflow Detection Logic

- **Carry into MSB \oplus Carry out of MSB**
 - For a N-bit ALU: $\text{Overflow} = \text{CarryIn}[N-1] \text{ XOR } \text{CarryOut}[N-1]$

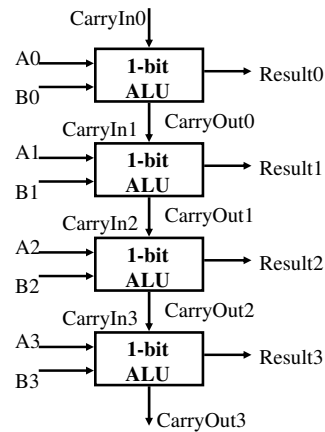


COMP3211/9211

2011S1 wk3_1 P16

What about performance?

- **Critical Path of n-bit Ripple-carry adder is $n \cdot T_d$**
 - Time complexity: $O(n)$



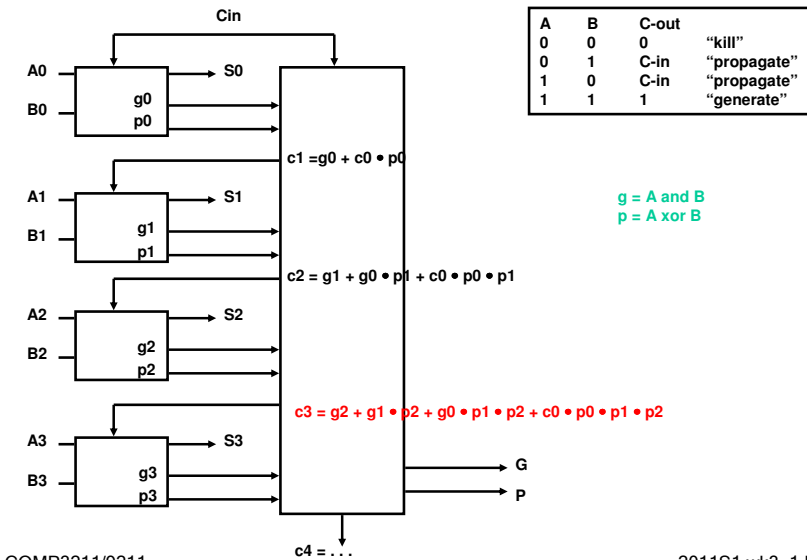
- **Design Trick 4: throw hardware at it**

COMP3211/9211

2011S1 wk3_1 P17

Carry Look Ahead

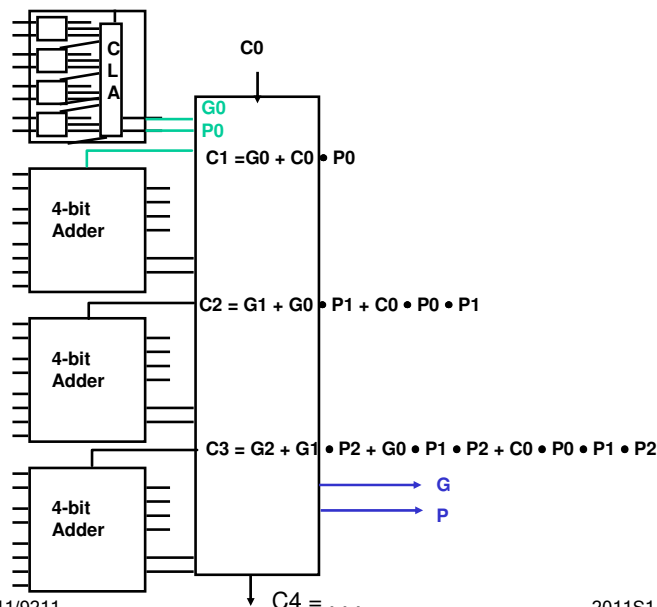
- **Design trick 5: peek**



COMP3211/9211

2011S1 wk3_1 P18

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

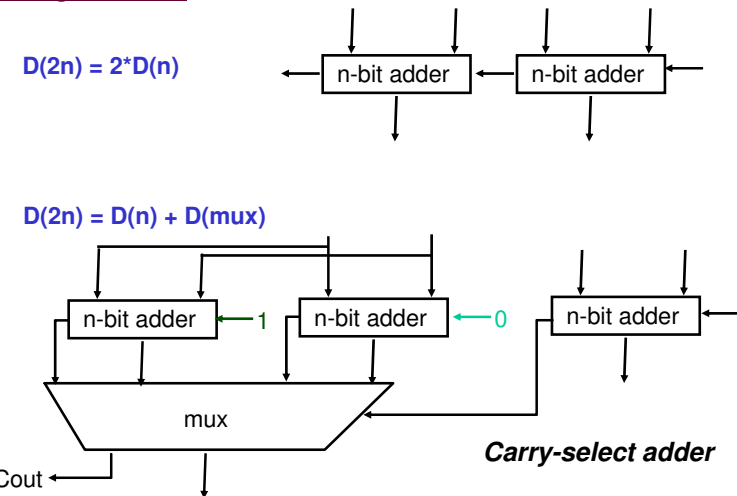


COMP3211/9211

2011S1 wk3_1 P19

Carry-select Adder

- **Design Trick 6: Guess**

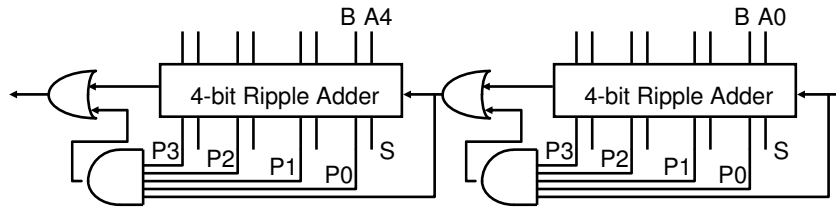


COMP3211/9211

2011S1 wk3_1 P20

Carry Skip Adder

- **Design Trick 7** : reduce worst case delay



Just speed up the slowest case for each block

Multiply (unsigned)

- **Paper and pencil example (unsigned):**

```

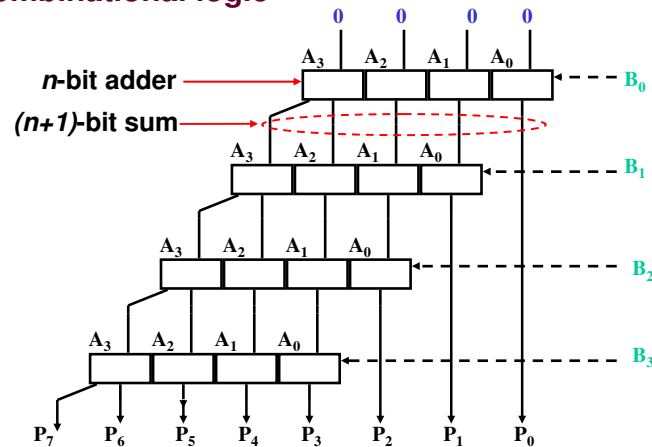
Multiplicand   1000
Multiplier     1001
-----
                1000
               0000
              0000
             1000
-----
Product       01001000
    
```

Implement this process!

- **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)
- **Four versions of design:**
 - successive refinement

Unsigned Multiplier (version 0)

- **Combinational logic**



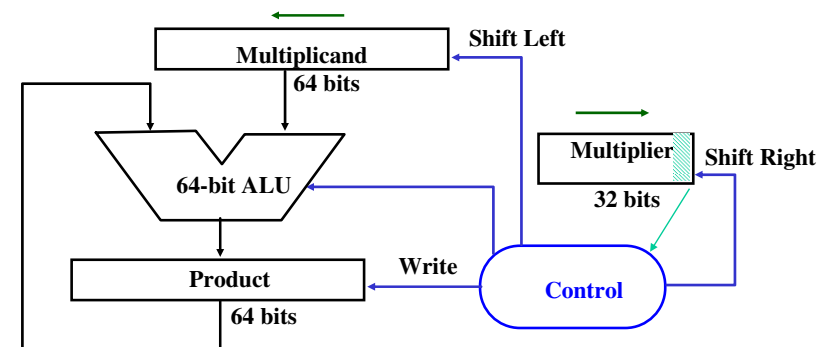
- Stage i accumulates $A \cdot 2^i$ if $B_i = 1$

- How much hardware for 32 bit multiplier? Critical path length (maximum delay on any path through circuit)?

Unsigned Multiplier (version 1)

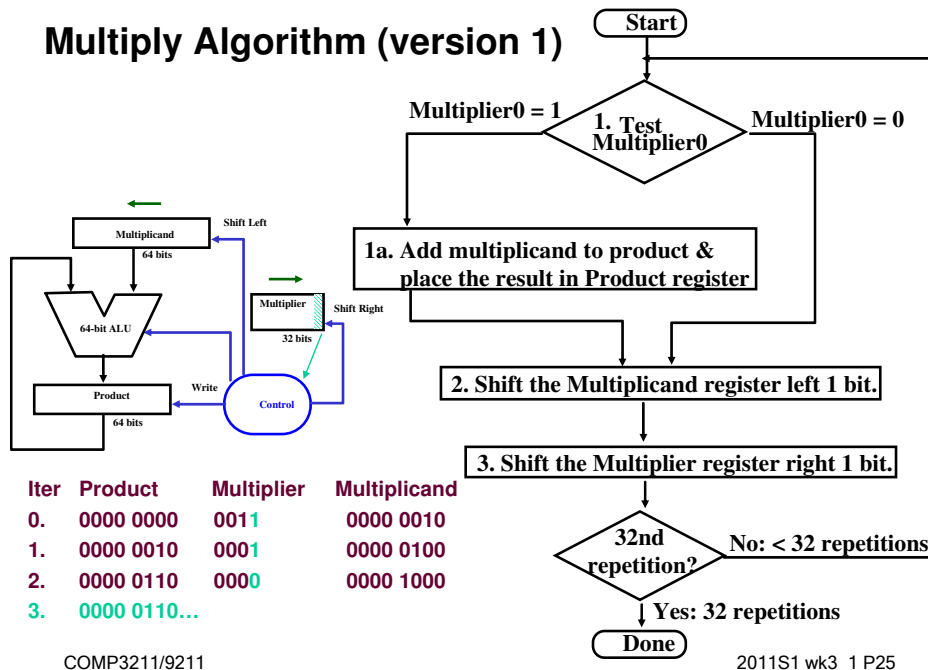
- **Sequential logic**

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



Multiplier = datapath + control

Multiply Algorithm (version 1)



Observations on Multiply Version 1

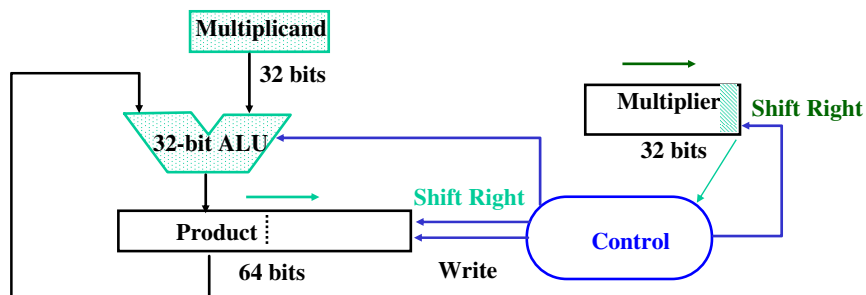
- 1 clock per bit $\Rightarrow O(n)$ for n -bit multiply
- 1/2 bits in multiplicand register always 0
 - 0's inserted in the right of multiplicand register as left-shifted
- least significant bits of product never changed once formed
 - Only 32 bits are effectively involved in the addition each time.
- 64-bit adder is wasteful
- Instead of shifting multiplicand to left, shift product to right?

COMP3211/9211

2011S1 wk3_1 P26

Unsigned Multiplier (version 2)

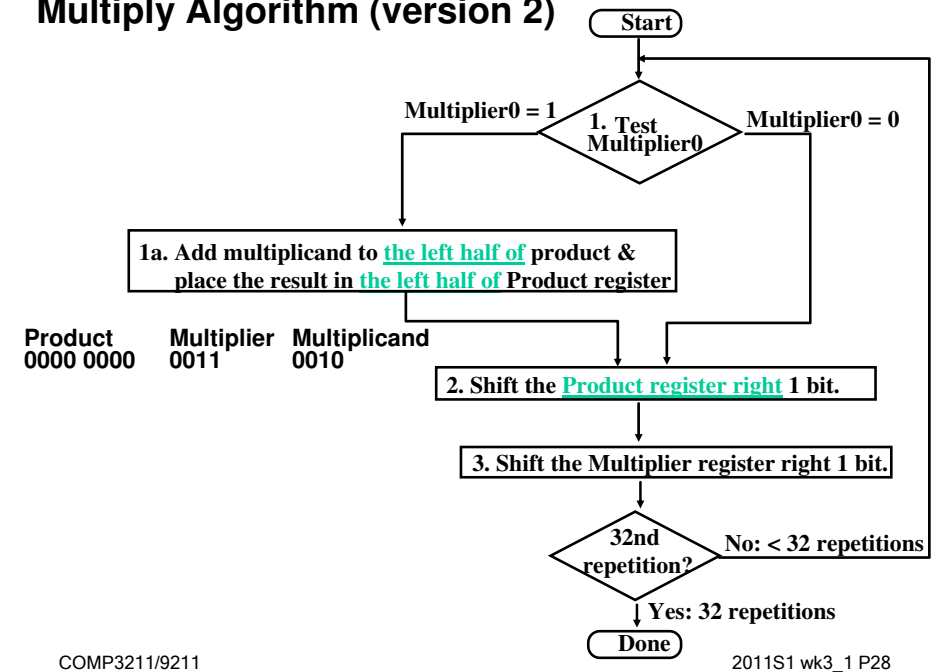
- 32-bit Multiplicand reg, 32-bit ALU, 64-bit Product reg, 32-bit Multiplier reg



COMP3211/9211

2011S1 wk3_1 P27

Multiply Algorithm (version 2)

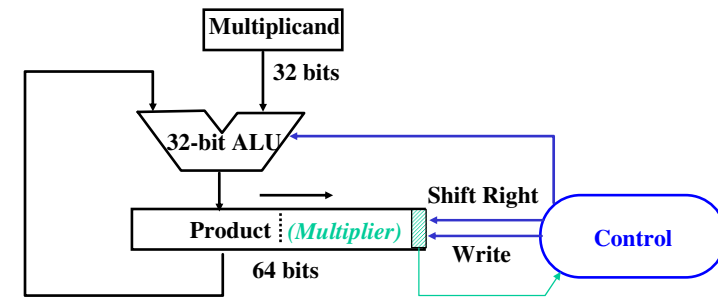


Observations on Multiply Version 2

- **Product register wastes space that exactly matches size of multiplier**
 - combine Multiplier register and Product register

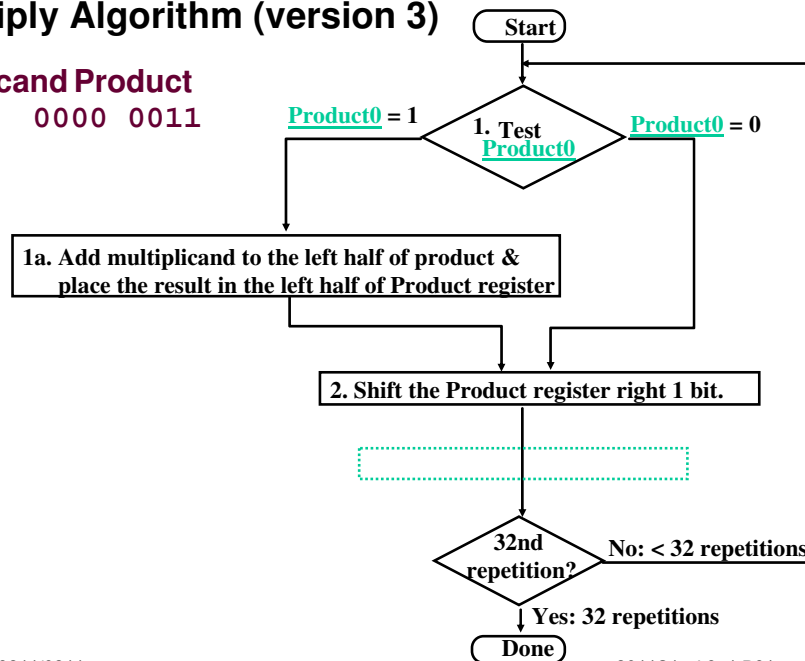
Unsigned Multiplier (version 3)

- **32-bit Multiplicand reg, 32-bit ALU, 64-bit Product reg, (0-bit Multiplier reg)**



Multiply Algorithm (version 3)

Multiplicand Product
0010 0000 0011



Observations on Multiply Version 3

- **2 steps per bit because Multiplier & Product combined**
- **MIPS registers Hi and Lo are left and right half of Product register**
- **The multiplier is for unsigned multiplication. What about signed multiplication?**
 - Booth's Algorithm is an elegant way to multiply signed numbers using same hardware as before and save cycles
 - can handle multiple bits at a time

Motivation for Booth's Algorithm

Example $2 \times 6 = 0010 \times 0110$:

```

      0010
x     0110
+-----
+    0000      shift (0 in multiplier)
+    0010      add (1 in multiplier)
+    0010      add (1 in multiplier)
+    0000      shift (0 in multiplier)
+-----
00001100
    
```

• ALU with add or subtract gets same result in more than one way:

```

6      = - 2 + 8
0110   = - 00010 + 01000 = 11110 + 01000
    
```

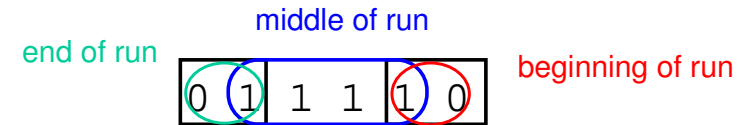
• For example

```

      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

Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one



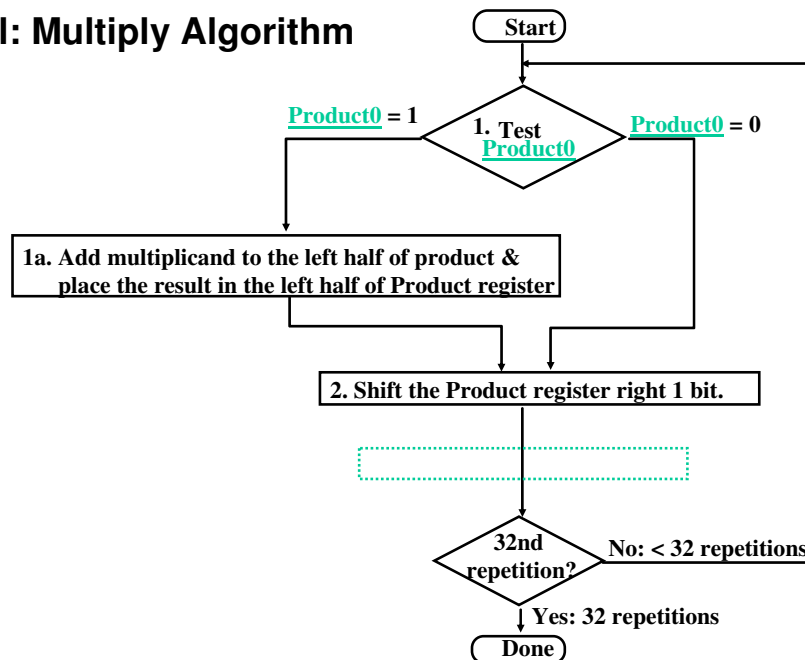
Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	000111 <u>1</u> 000	sub
1	1	Middle of run of 1s	00011 <u>11</u> 000	none
0	1	End of run of 1s	00 <u>0</u> 1111000	add
0	0	Middle of run of 0s	00 <u>00</u> 1111000	none

Originally for Speed (when shift was faster than add)

```

-1
+ 1000
01111
    
```

Recall: Multiply Algorithm



Booths Example (2 x 7)

Operation	Multiplicand	Product	Current bit	Bit to right next?
0. initial value	0010	0000 0111 0	↓	10 -> sub
1a. $P = P - m$	1110	+ 1110 1110 0111 0	↓	shift P (sign ext)
1b.	0010	1111 0011 1	↓	11 -> nop, shift
2.	0010	1111 1001 1	↓	11 -> nop, shift
3.	0010	1111 1100 1	↓	01 -> add
4a.	0010	+ 0010 0001 1100 1	↓	shift
4b.	0010	0000 1110 0	↓	done

Booths Example (2 x -3)

Operation	Multiplicand	Product	Current bit Bit to right next?
0. initial value	0010	0000 1101 0	10 -> sub
1a. $P = P - m$	1110	+1110 1110 1101 0	shift P (sign ext)
1b.	0010	1111 0110 1 + 0010	01 -> add
2a.		0001 0110 1	shift P
2b.	0010	0000 1011 0 + 1110	10 -> sub
3a.	0010	1110 1011 0	shift
3b.	0010	1111 0101 1	11 -> nop
4a		1111 0101 1	shift
4b.	0010	1111 1010 1	done

COMP3211/9211

2011S1 wk3_1 P37

Wallace Tree*

- An efficient hardware implementation of a digital circuit that multiplies two integers.
- Has $O(\log n)$ complexity for $n \times n$ two integer complications
- Three steps
 - Bit multiply (AND)
 - Reduce the bit-wise sum to multiple layers of HA or FA
 - Each leave FA/HA producing one-bit sum and 1-bit carry, two wires
 - Group the wires from the HA/FAs into two numbers, and add them with a conventional adder.

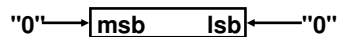
COMP3211/9211

2011S1 wk3_1 P38

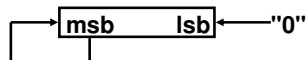
Shifters

• Two kinds:

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



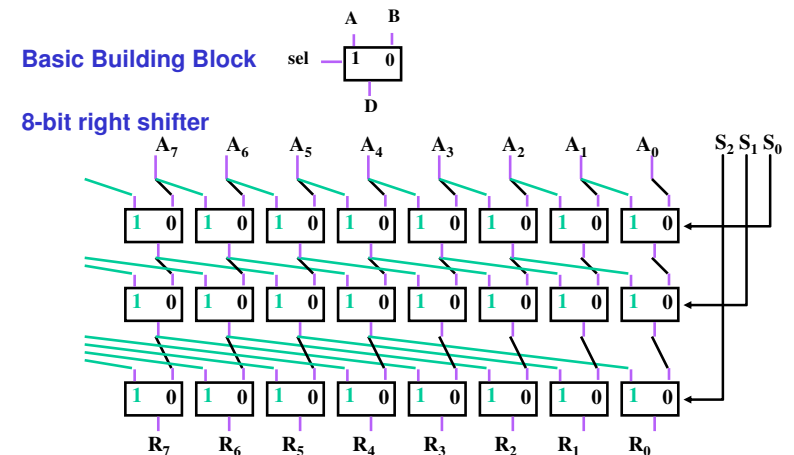
- arithmetic-- on right shifts, sign extend



COMP3211/9211

2011S1 wk3_1 P39

Combinational Shifter from MUXes



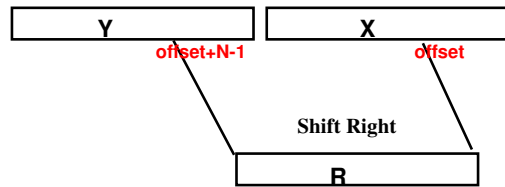
- What comes in the MSBs?
- How many levels for 32-bit shifter?
- What if we use 4-1 Muxes ?

COMP3211/9211

2011S1 wk3_1 P40

Funnel Shifter*

- A funnel shifter can do all three types shifts
- Selects N -bit R from $2N$ -bit inputs Y and X
 - Shift by k bits ($0 \leq k < N$)



- **Example**

- Right shift input A by k bits

Logical: $Y=0, X=A, \text{offset}=k$

Arithmetic?

Rotate?

Left shifts?

