

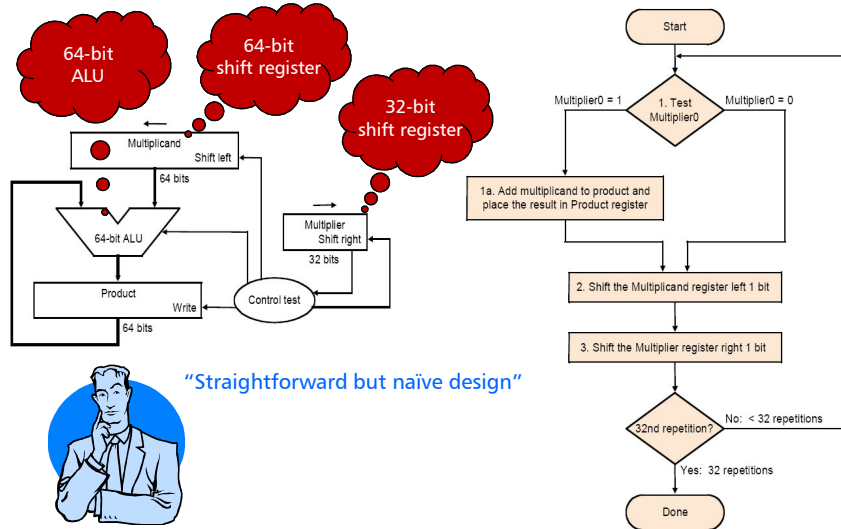
Multiplication

- More complicated than addition
 - A straightforward implementation will involve *addition* and *shifting*
- A “more complex operation” implies
 - More area (on silicon) and/or
 - More time (more clock cycles or longer clock cycle time)
- Let’s begin from a simple, straightforward method
 - For now, consider only unsigned numbers!

Straightforward algorithm

```
  01010010 (multiplicand)
x  01101101 (multiplier)
-----
```

Hardware design 1



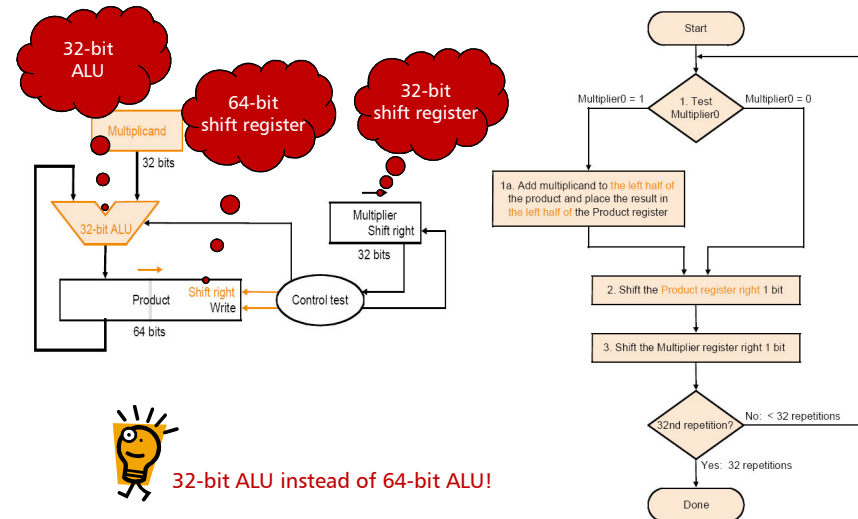
"Straightforward but naïve design"

CS/CoE0447: Computer Organization and Assembly Language

University of Pittsburgh

40

Hardware design 2



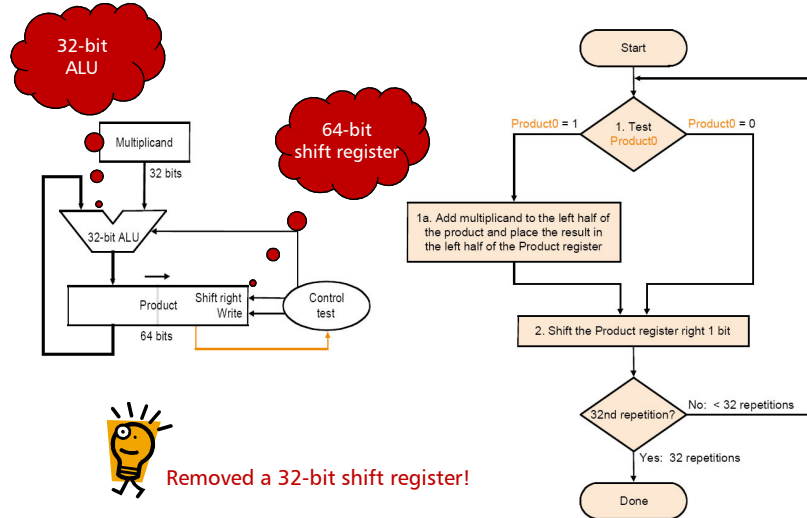
32-bit ALU instead of 64-bit ALU!

CS/CoE0447: Computer Organization and Assembly Language

University of Pittsburgh

41

Hardware design 3



CS/CoE0447: Computer Organization and Assembly Language

University of Pittsburgh

42

Example

- Let's do 0010×0110 (2×6), unsigned

Iteration	Multiplicand	Implementation 3	
		Step	Product
0	0010	initial values	0000 0110
1	0010	1: 0 -> no op	0000 0110
		2: shift right	0000 0011
2	0010	1: 1 -> product = product + multiplicand	0010 0011
		2: shift right	0001 0001
3	0010	1: 1 -> product = product + multiplicand	0011 0001
		2: shift right	0001 1000
4	0010	1: 0 -> no op	0001 1000
		2: shift right	0000 1100

CS/CoE0447: Computer Organization and Assembly Language

University of Pittsburgh

43

Booth's encoding

- Three symbols to represent a binary number: {1,0,-1}
- Examples (8-bit encoding)
 - -1
 - 11111111 (two's complement)
 - 0000000-1 (Booth's encoding)
 - 14
 - 00001110 (two's complement)
 - 000100-10 (Booth's encoding)
- Bit transitions in number (in two's complement encoding) show how Booth's encoding works
 - 0 to 0 (from right to left): 0
 - 0 to 1: -1
 - 1 to 1: 0
 - 1 to 0: 1

Booth's encoding

- Key point
 - A "1" in the multiplier implies an addition operation
 - If you have many "1"s – that means many addition operations
- Booth's encoding is useful because it can reduce the number of addition operations you have to perform
- With Booth's encoding, partial results are obtained by
 - Adding multiplicand
 - Adding 0
 - Subtracting multiplicand

Booth's algorithm in action

- Let's do 0010×1101 (2×-3)

Iteration	Multiplicand	Booth's algorithm	
		Step	Product
0	0010	initial values	0000 1101 0
1	0010	10 -> product = product - multiplicand	1110 1101 0
		shift right	1111 0110 1
2	0010	01 -> product = product + multiplicand	0001 0110 1
		shift right	0000 1011 0
3	0010	10 -> product = product - multiplicand	1110 1011 0
		shift right	1111 0101 1
4	0010	11 -> no op	1111 0101 1
		shift right	1111 1010 1