Chapter 3 – part 1

Arithmetic for Computers

- Addition
- Subtraction
- Multiplication

Arithmetic for Computers

- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Integer Addition

- Example: 7 + 6

		(0)		(0)		(1)		(1)		(0)		(Carries)
. . .		0		0		0		1		1		1
. . .		0		0		0		1		1		0
. . .	(0)	0	(0)	0	(0)	1	(1)	1	(1)	0	(0)	1

- Overflow if result out of range
  - Adding "+" and "-" operands, no overflow
  - Adding two "+" operands
    - Overflow if result sign is 1
  - Adding two "-" operands
    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

+7:	0000 0000 … 0000 0111
–6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- Overflow if result out of range
  - Subtracting two “+” or two “-” operands, no overflow
  - Subtracting “+” from “-” operand
    - Overflow if result sign is 0
  - Subtracting “-” from “-” operand
    - Overflow if result sign is 1

Dealing with Overflow

- Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui`, `subu` instructions
- Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi`, `sub` instructions
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register
    - Jump to predefined handler address
    - `mfc0` (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action

Arithmetic for Multimedia

- Graphics and media processing operates on vectors of 8-bit and 16-bit data
  - Use 64-bit adder, with partitioned carry chain
    - Operate on 8×8-bit, 4×16-bit, or 2×32-bit vectors
  - SIMD (single-instruction, multiple-data)
- Saturating operations
  - On overflow, result is largest representable value
    - c.f. 2s-complement modulo arithmetic
  - E.g., clipping in audio, saturation in video

Multiplication is vexation, Division is as bad;
The rule of three doth puzzle me,
And practice drives me mad.

Anonymous,
Elizabethan manuscript, 1570

Long hand multiplication of two binary numbers.

$$
\begin{array}{r cccccc l}
 & & & & 1 & 0 & 1 & A \\
 & & & \times & 1 & 1 & 0 & B \\
\cline{5-7}
 & & & & 0 & 0 & 0 & \\
\text{partial product} \rightarrow & & & 1 & 0 & 1 & & \\
\text{partial product} \rightarrow & & 1 & 0 & 1 & & & \\
\cline{2-7}
 & 0 & 1 & 1 & 1 & 1 & 0 & A \times B
\end{array}
$$

note that at each addition location a one bit full binary adder will work.

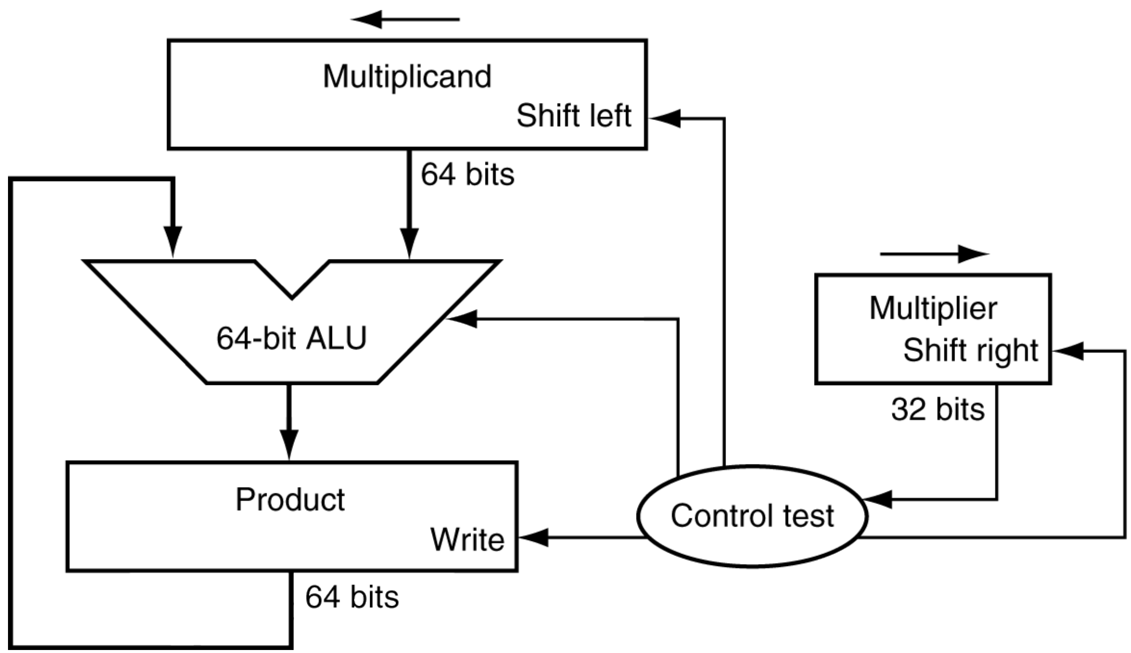
Multiplication

- Start with long-multiplication approach

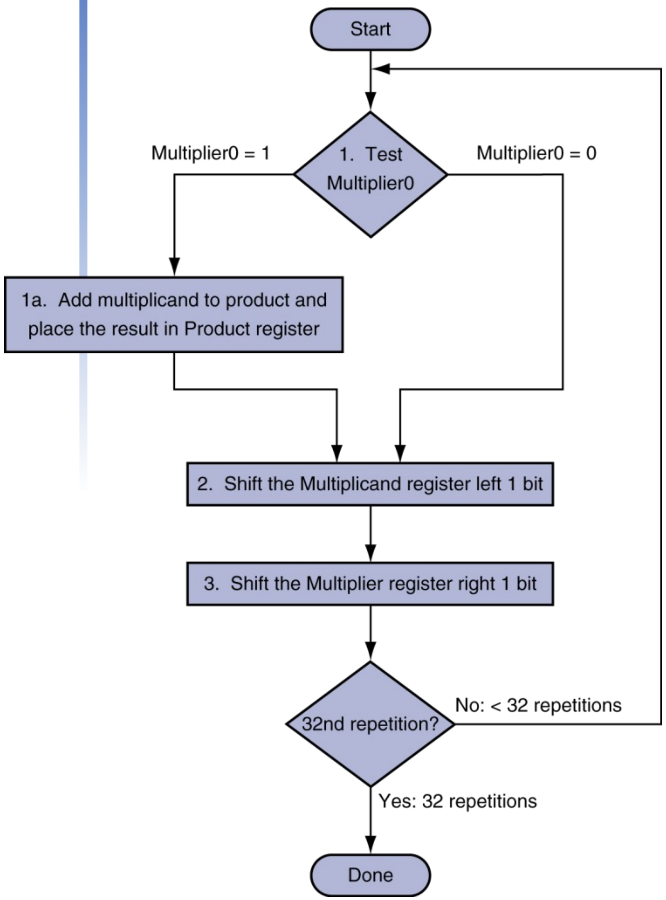
multiplicand
multiplier
product

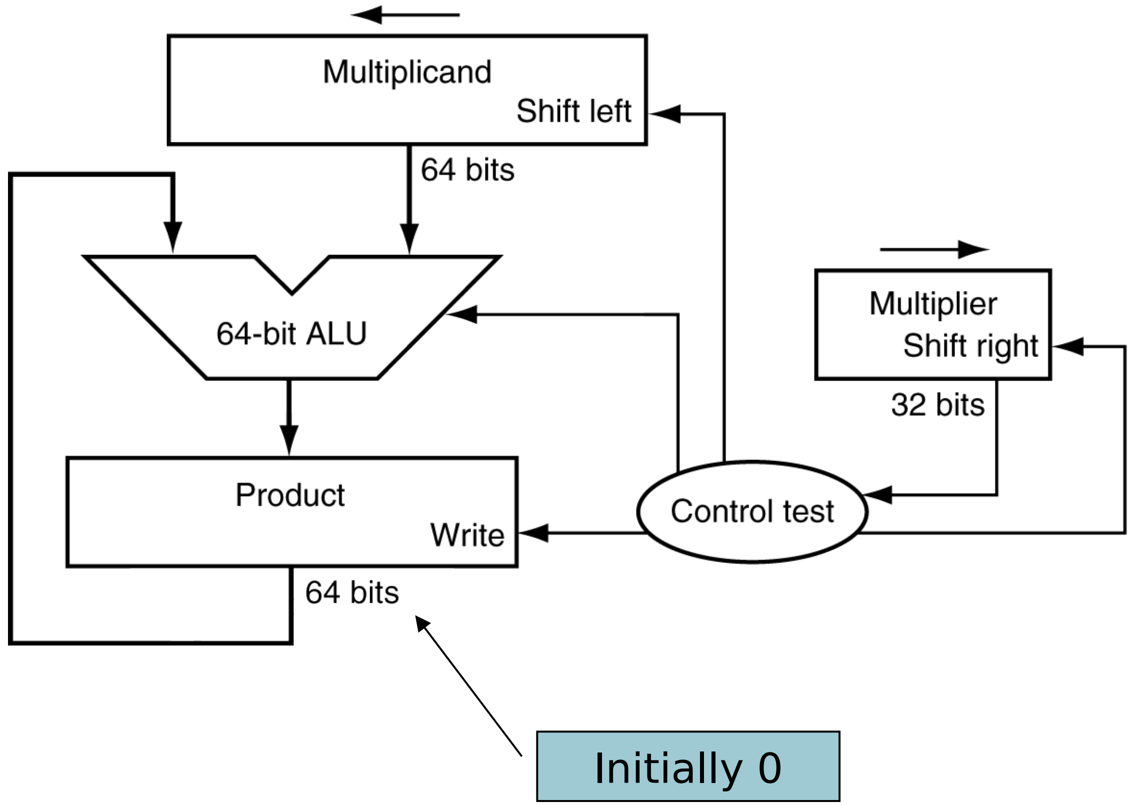
```
     1000
  ×  1001
  -------
     1000
    0000
   0000
  1000
  -------
  1001000
```

Length of product is the sum of operand lengths



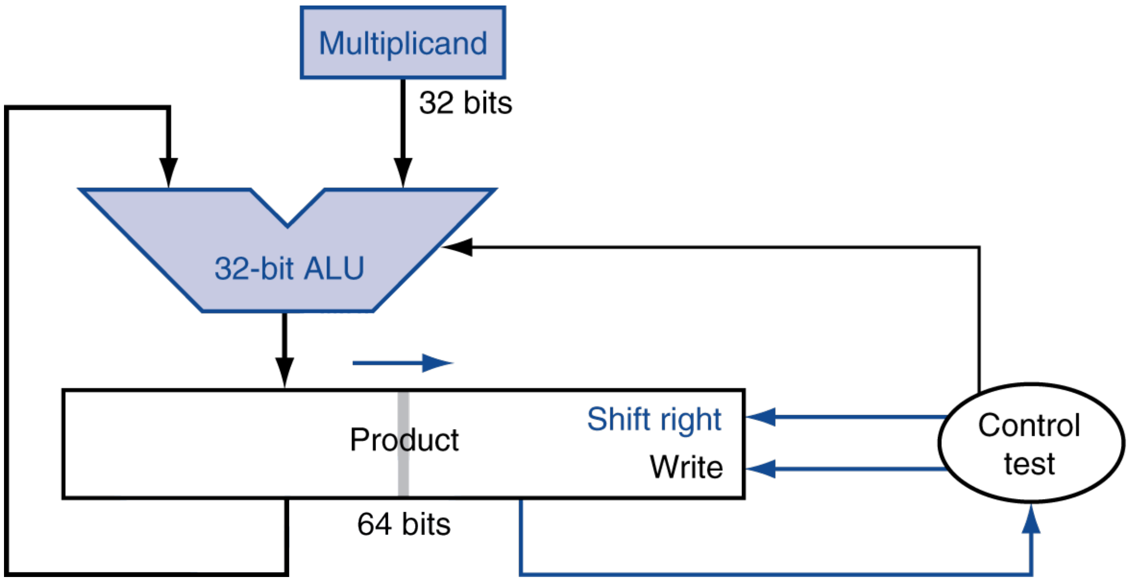
Multiplication Hardware





Optimized Multiplier

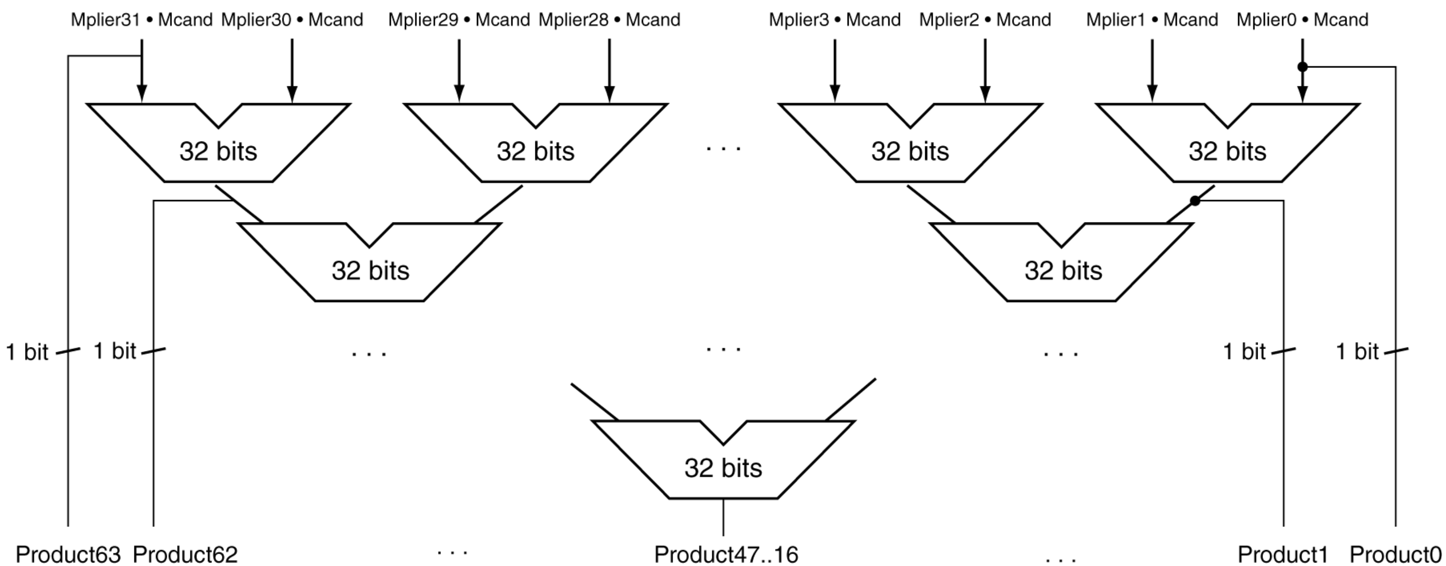
- Perform steps in parallel: add/shift



- One cycle per partial-product addition
  - That's ok, if frequency of multiplications is low

Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



- Can be pipelined
  - Several multiplication performed in parallel

MIPS Multiplication

- Two 32-bit registers for product
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instructions
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd