Arithmetic for Computers

COMPUTER ORGANIZATION AND ARCHITECTURE

Arithmetic for Computers

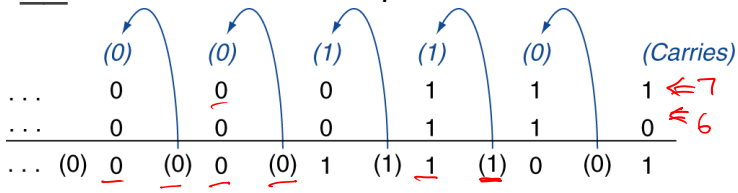
- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Integer Addition

- Overflow if result out of range
  - Adding +ve and –ve operands, overflow? No
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

0111 ~ 1111
0111 ~ 1111 (+
1 ~~

Example: 7 + 6



Integer Subtraction

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

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

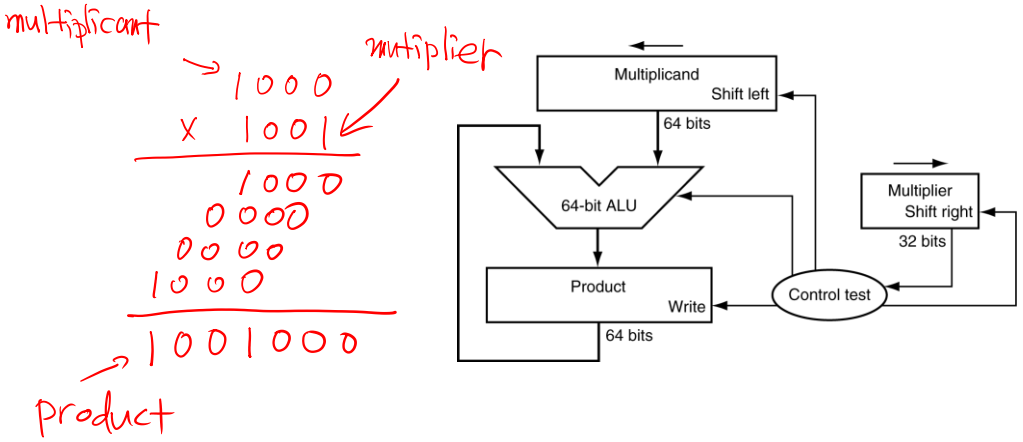
- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, overflow? NO
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

Dealing with Overflow

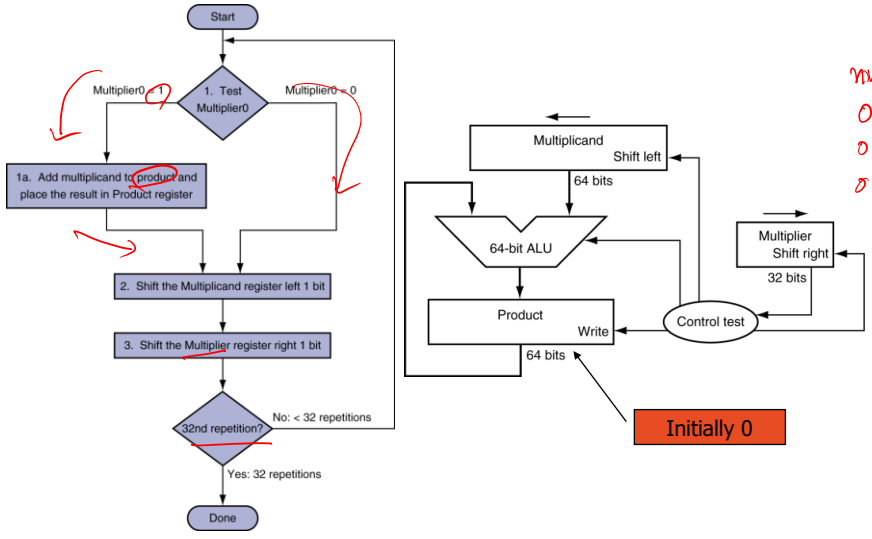
- Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui`, `subu` instructions
- Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi`, `sub` instructions
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register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

Multiplication

- Start with long-multiplication approach



Multiplication Hardware



2 x 3

0010 0011

multiplier multiplicand prod

0011 0000 0010 0000 0000

0001 0000 0100 0000 0010

0000 0000 1000 0000 0110

0001 0000 0000 0110

0010

0100

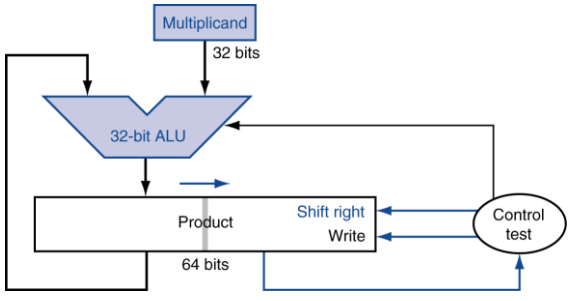
1000

0000

6.

Optimized Multiplier

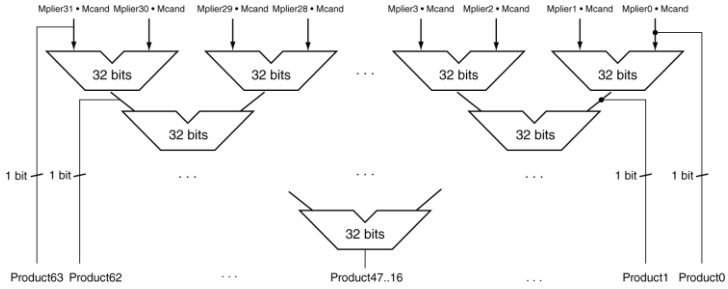
- Perform steps in parallel: add/shift



- One cycle per partial-product addition

Faster Multiplier

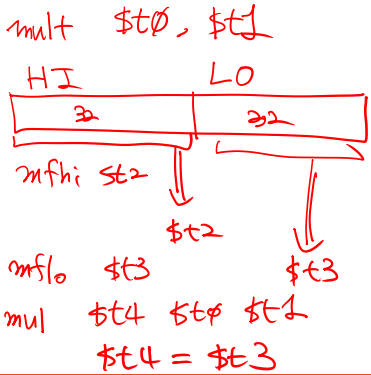
- Uses multiple adders
  - Cost/performance tradeoff



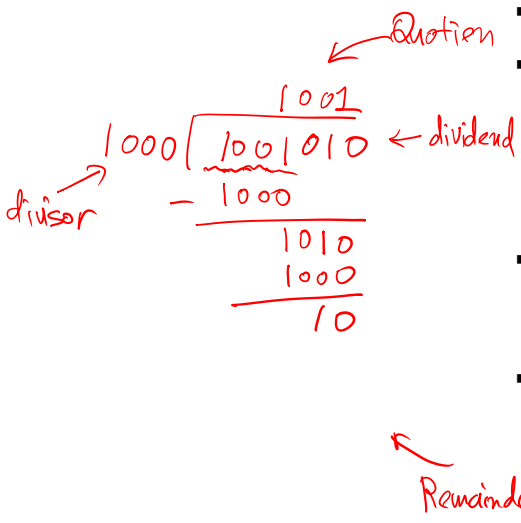
- Can be pipelined
  - Several multiplication performed in parallel

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



Division



- Check for 0 divisor
- Long division approach
  - If divisor ≤ dividend bits
    - 1 bit in quotient, subtract
  - Otherwise
    - 0 bit in quotient, bring down next dividend bit
- Restoring division
  - Do the subtract, and if remainder goes < 0, add divisor back
- Signed division
  - Divide using absolute values
  - Adjust sign of quotient and remainder as required

11/9/26