# Exam 2 review

Monday, April 8, 2024 1:00 PM

Chapter 3
Sections 3.1 - 3.6

Addition and subtraction will be the same thing. Be careful with overflow and underflow!

**Multiplication**
Start with long-multiplication approach

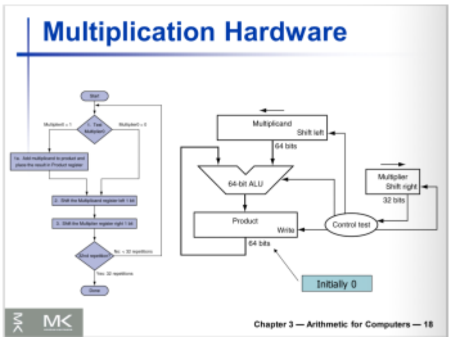
The normal approach: result is the sum of operand lengths

Binary multiplication is: bunch of shifts and adds

Note that: n-bit number multiply with n-bit number will result in 2n-bit number

**Integer multiplication**
Key elements
1. Examine multiplier bits from right to left
2. Shift multiplicand left one position each step
3. Simplification: each step, add multiplicand to running product total, but only if multiplier bit == 1

Detail step:
1. Initialize product register to 0
2. Get result of the multiplier bit then add it to product (Ignore if bit is 0)
3. Shift multiplicand left
4. Move to the next one



## Multiplication Hardware

Chapter 3 — Arithmetic for Computers — 18

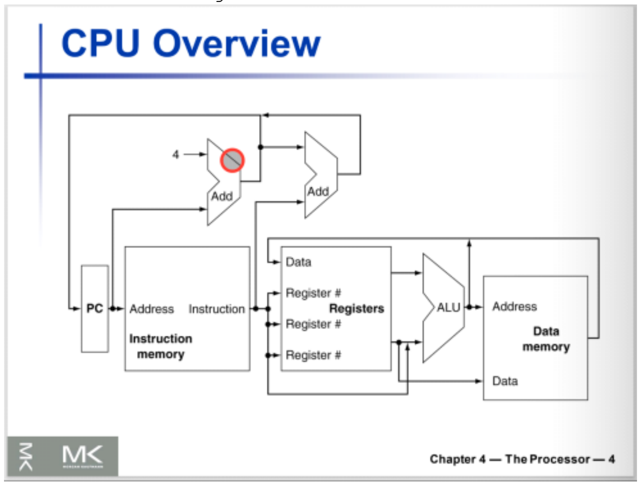
Can optimize this by parallel operation add and/shift

**LEGv8 multiplication**
Three multiply instructions:
- MUL: multiply
  - Gives the lower 64 bits of the product
- SMULH: Sign multiply high
  - Gives the upper 64 bits of the product, assuming the operands are signed
- UMULH: unsigned multiply high
  - Gives the lower 64 bits of the product, assuming the operands are unsigned

**Floating point**
Follow normal instruction and rmb some

Chapter 4
Read all except 4.6 to 4.9 and 4.12

**Sequence for Instruction Execution**
- PC -> Instruction memory, fetch instruction
- Register numbers -> register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC <- target address or PC + 4

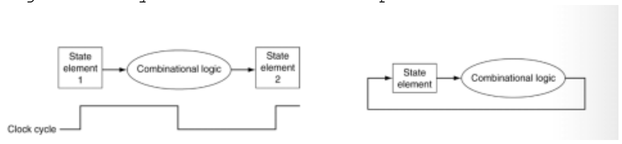


## CPU Overview

Chapter 4 — The Processor — 4

- To join wires, use mutiplexers (MUX)
- Add control to control the sequence of activation for every components

**Clocking Methodology**
- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period



**Data-path**
Elements that process data and addresses in the CPU

**R-format instructions**
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

Using: Register file (R1, R2) and ALU

**Load/store instructions**
- Read register operand
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: read memory and update register
- Store write register value to memory

**Branch instructions**
- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement

are unsigned

**Floating point**
Follow normal instruction and rmb some special case such as infinity, overflow and underflow

IEEE floating-point format

Bias table

| | Single |
|---|---|
| Exponent | 8 |
| Fraction | 23 |
| Bias | 127 |

Rmb how to normalize and denormalize numbers as well

Denormalize usually work for underflow number

Special cases:

| A | Exp | Fraction |
|---|---|---|
| Infinities | All 1 | All 0 |
| NaNs | All 1 | Not all 0 |

Learn again about how to add two floating-point

?Rounding with guard bit

**Subword parallelism**
Graphics and audio application can take advantage of performing simultaneous operations on short vectors
Example: 128-bit adder can perform in
    Sixteen 8-bit adds
    Eight 16-bits adds
    Four 32-bit adds

Also called data-level parallelism, vector parallelism, or Single Instruction, Multiple Data (SIMD)

**Fallacies and pitfalls**
Generally stem from the difference between the limited precision of computer arithmetic and the unlimited precision of natural arithmetic.

*Fallacy: Just as a left shift instruction can replace an integer multiply by a power of 2, a right shift is the same as an integer division by a power of 2*

=> Shift -> mul/div by 2^x => But only work with unsigned num

*Pitfall: Floating-point addition is not associative*

When adding two large number of opposite signs plus a small number
C = -1.5 * 10^30
A = 1.5 * 10^30
B = 1

C + (A + B) = -1.5 * 10^30 + (1.5*10^30 + 1.0) = 0

(C + A) + B = (-1.5*10^30 + 1.5*10^30) + 1.0 = 1

NOPE THEY ARE NOT THE SAME. EVEN THO

- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

**Pipelining**
Literally, overlapping execution.
Pipelining will get the longest time for a component to process the data and use it as the time for all other processes.

LEGv8 Pipeline
1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: execute operation or calculate address
4. MEM: access memory operand
5. WB: Write result back to register

**4.10. Parallelism via Instructions**
NOTICE: complex topics

?????

**4.14. Fallacies and pitfalls**

*Fallacy: pipelining is easy*

*Fallacy: Pipelining ideas can be implemented independent of technology*
-> Today, power are leading to less aggressive and more efficient designs

*Pitfall: failure to consider instruction set design can adversely impact pipelining*

**4.15. Concluding remarks**

THEY SHOULD BE

*Fallacy: parallel execution strategies that work for integer data types also work for floating-point data types*

*Fallacy: only theoretical mathematician care about floating-point accuracy*

**3.11 Concluding remarks**

- Bits have no inherent meaning
    - Interpretation depends on the instructions applied
- Computer representations of numbers
    - Finite range and precision
    - Need to account for this in programs
- ISAs support arithmetic
    - Signed and unsigned integers
    - Floating-point approximation to reals
- Bounded range and precision
    - Operations can overflow and underflow