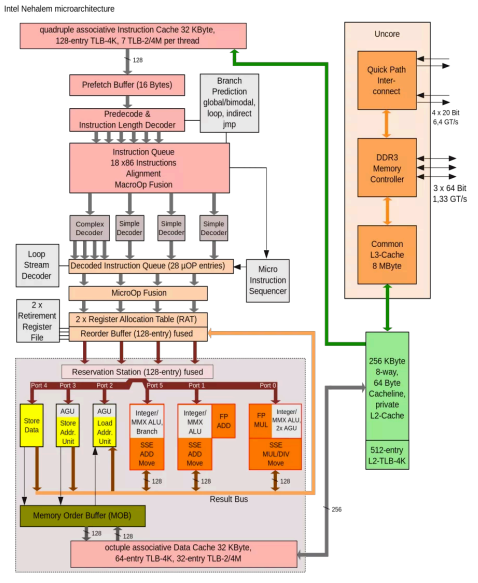
CSC 411

Computer Organization (Spring 2022)
Lecture 12: Arithmetic Operations

Prof. Marco Alvarez, University of Rhode Island

Nehalem /nəˈheɪləm/[1] is the codename for an Intel processor microarchitecture released in November 2008.[2] Nehalem was used in the first generation of the Intel Core processors (Core i7 and i5, with Core i3 being based on the subsequent Westmere and Sandy Bridge designs). Nehalem is the successor to the older Core microarchitecture (Intel Core 2 processors).[3]



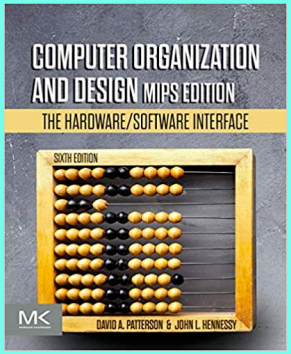
https://www.pcgamer.com/au/how-processors-work/

Disclaimer

Some of the following slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface

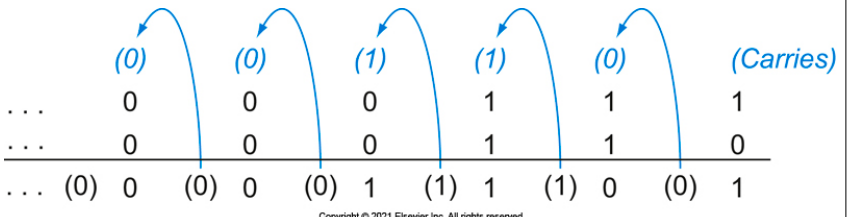


Addition/Subtraction

Topics

- Operations on integers
  - addition and subtraction
  - multiplication and division
  - dealing with overflow

Integer addition



$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten} \end{array}$$

Integer addition

- Overflow if result out of range
  - adding positive and negative operands, no overflow
  - adding two positive operands
    - overflow if result's most significant bit is 1
  - adding two negative operands
    - overflow if result's most significant bit is 0

e.g. using 8 bits perform 57+80

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

Integer subtraction

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

- Add negation of second operand
  - Two's complement negation
    - invert all bits and add 1 (special cases — negate smallest negative)

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

Integer subtraction

- Overflow if result out of range
  - subtracting two positive or two negative operands, no overflow
  - subtracting positive from negative operand
    - overflow if result's most significant bit is 0
  - subtracting negative from positive operand
    - overflow if result's most significant bit is 1

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

SIMD instructions

Multiplication

Warming-up

Multiplicand 1000_{ten}

Multiplier x 1001_{ten}

1000

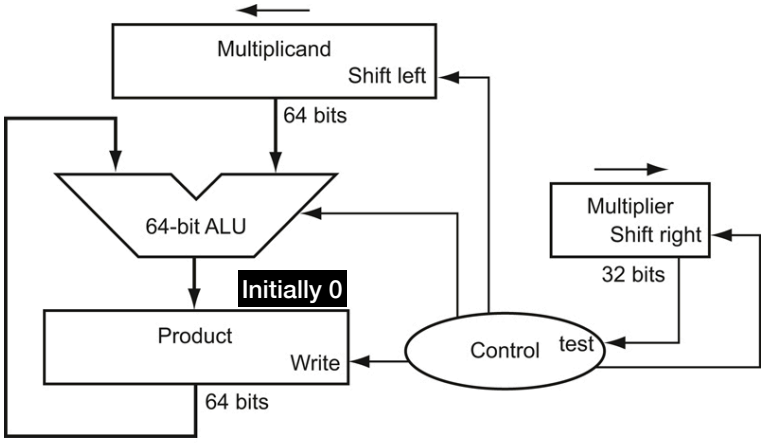
0000

0000

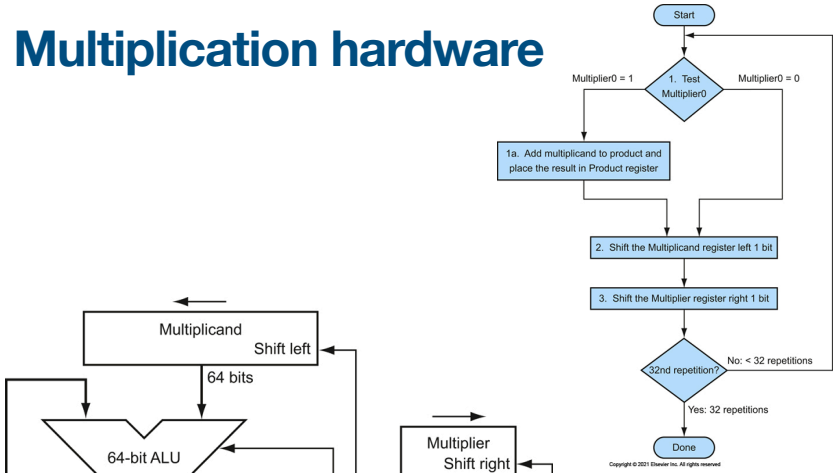
1000

Product 1001000_{ten}

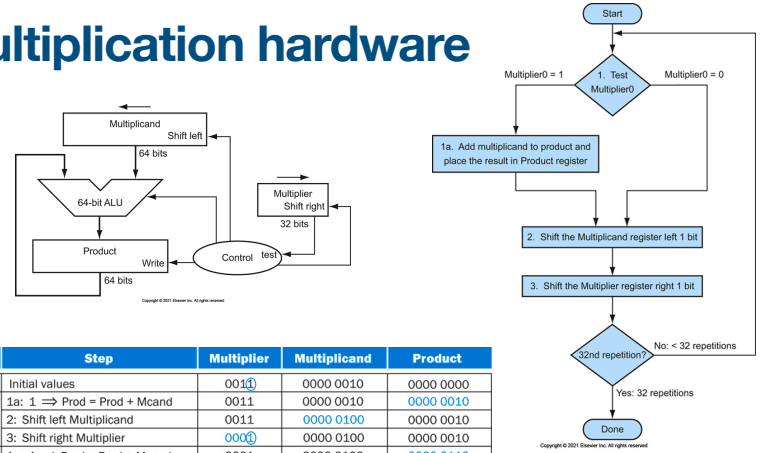
Multiplication



Multiplication hardware

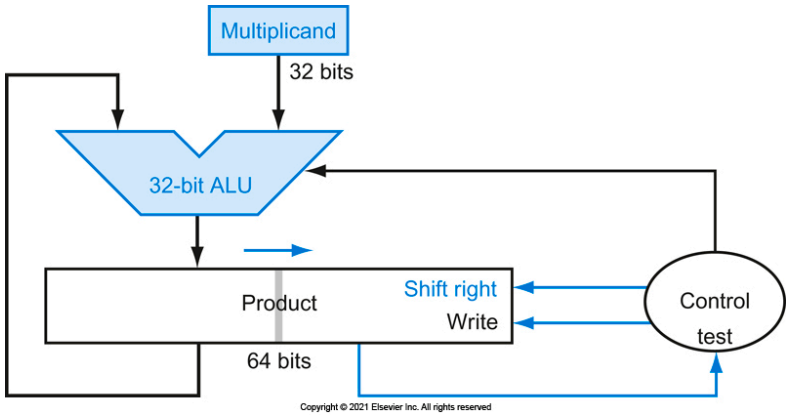


Multiplication hardware

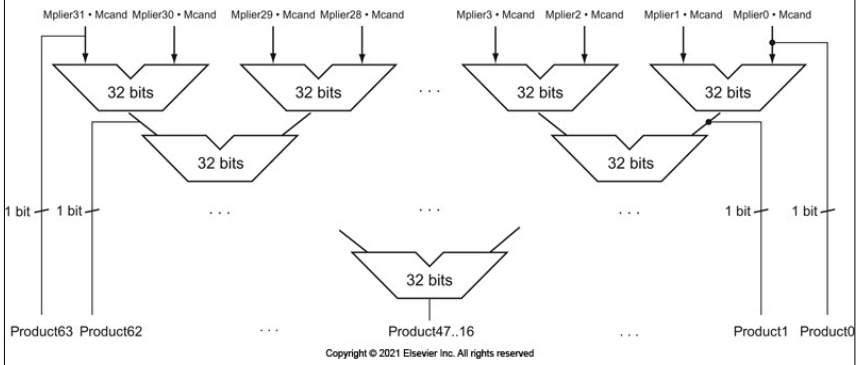


Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Better multiplication



Fast multiplication



Rather than use a single 32-bit adder 31 times, this hardware "unrolls the loop" to use 31 adders and then organizes them to minimize delay.

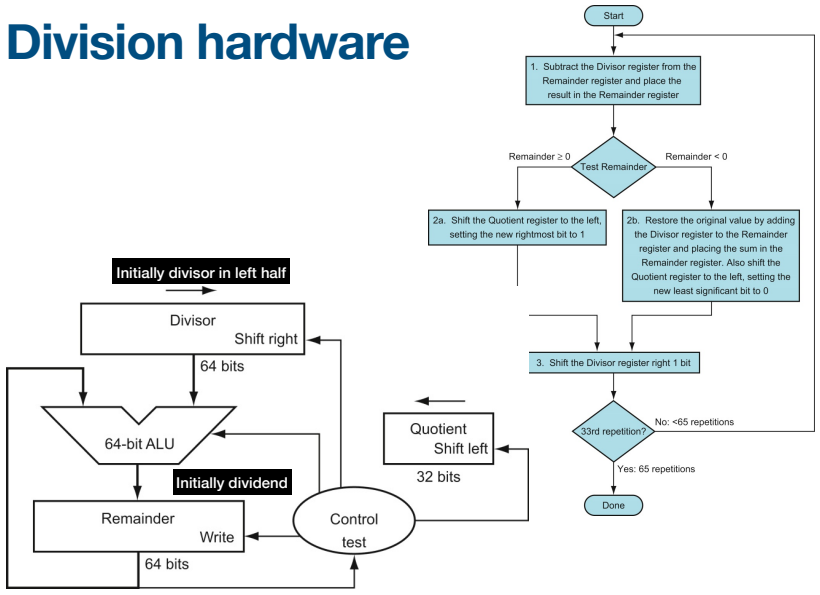
Division

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

quotient
1001
divisor 1000 | 1001010 dividend
-1000
10
101
1010
-1000
remainder 10

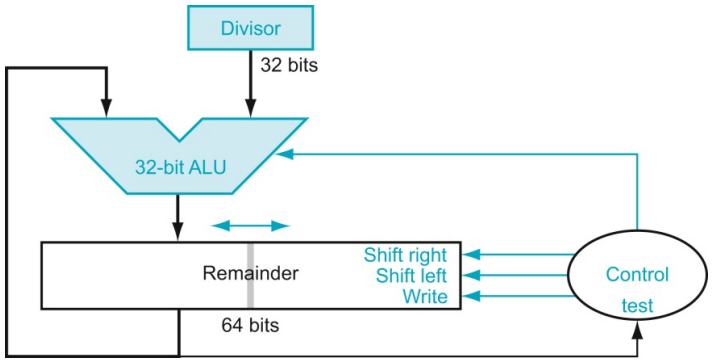
Division hardware



Division example

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 ⟹ +Div, SLL Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 ⟹ +Div, SLL Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 ⟹ +Div, SLL Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	⓪000 0011
	2a: Rem ≥ 0 ⟹ SLL Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	⓪000 0001
	2a: Rem ≥ 0 ⟹ SLL Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Improved division



-Very similar to a multiplier, in fact, same hardware can be used for both
-Can't use parallel hardware as in multiplier