CPS 104
Computer Organization and Programming
Lecture 9: Integer Arithmetic.

Robert Wagner

Overview of Today's Lecture:

- **Integer Multiplication and Division.**

Read Appendix B

Integer Multiplication

- **Product = Multiplicand x Multiplier**
- **Example: 0011 x 0101**

Multiplicand			0	0	1	1	
Multiplier			0	1	0	1	
			0	0	1	1	
		0	0	0	0	*0*	
	0	0	1	1	*0*	*0*	
0	0	0	0	*0*	*0*	*0*	
Product	0	0	0	1	1	1	1

Multiplication Algorithm #1

- From Right-Left:
  - If multiplier digit = 1: add (shifted) copy of multiplicand to result.
  - If multiplier digit = 0: add 0 to result. (do nothing)
- 32 steps when multiplier is 32-bit number.
- Example: $3_{10} \times 5_{10}$ or $0011_2 \times 0101_2$
  Product = 00001111_2
- Product has twice as many bits as operands

Multiplication Algorithm #1

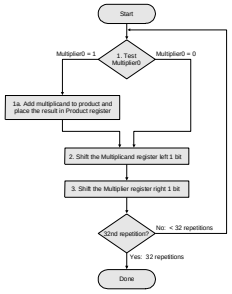


Figure Copyright Morgan Kaufmann

Multiplication Hardware #1

- Multiplicand starts in right half of register

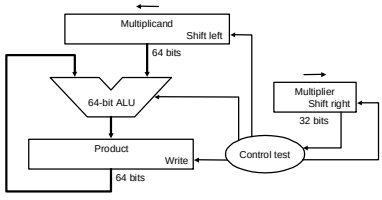
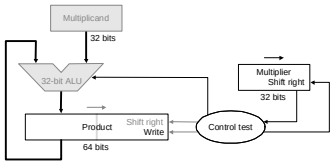


Figure Copyright Morgan Kaufmann

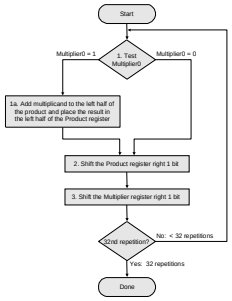
Multiplication Hardware #2

- Shift Multiplicand Left works same as Shift Product Right
- Only need 32 bits for multiplicand



- Possible to combine multiplier and product registers
  - Keep multiplier in low order 32 bits of product register

Multiplication Algorithm #2



Signed Multiplication II

- Overflow can occur during the addition step
  - Solution: Make the product register one bit longer
- Recall that the high-order bit of the multiplier has NEGATIVE weight
  - If the high-order bit of the multiplier is 1, SUBTRACT the multiplicand from the product, instead of adding it

Multiplication Examples (4 bit operands)

00111_2* 0111_2	11001_2* 1001_2	00111_2* 1001_2	11001_2* 0111_2
00000 0111	00000 1001	00000 1001	00000 0111
00111 0111	11001 1001	00111 1001	11001 0111
00011 1011	11100 1100	00011 1100	11100 1011
01010 1011	11110 0110	00001 1110	10101 1011
00101 0101	11111 0011	00000 1111	11010 1101
01100 0101	subtract	subtract	10011 1101
00110 0010	00110 0011	11001 1111	11001 1110
00011 0001	00011 0001	11100 1111	11100 1111
= 3*16+1	-7*-7 = 49	-(00011 0001)	-7*7 = -49

Booth Encoding

- Observation:
  - Can write number as difference of two numbers.
  - In particular: Can replace a string of 1s with initial subtract when we see a 1, and then an add when we see the bit AFTER the last 1
- Example 1: 7_{10}
  - $7_{10} = -1_{10} + 8_{10}$
  - $0111_2 = -0001_2 + 1000_2$
- Example 2: $110_{10} = 01101110_2$
  - $110_{10} = (-2_{10} + 16_{10}) + (-32_{10} + 128_{10})$
  - $01101110_2 = (-00000010_2 + 00010000_2) + (-00100000_2 + 10000000_2)$
- Works for signed numbers as well!

Booth's Algorithm

- Similar to previous multiply algorithm.
- (Current, Previous) bits of Multiplier:
  - 0,0: middle of string of 0s; do nothing
  - 0,1: end of a string of 1s; add multiplicand
  - 1,0; start of string of 1s; subtract multiplicand
  - 1,1: middle of string of 1s; do nothing
- Shift Product/Multiplier right 1 bit (as before)

Booth Multiplication Examples (4 bit operands)

0111_2* 0111_2	1001_2* 1001_2	0111_2* 1001_2	1001_2* 0111_2
0000 0111 0 -	0000 1001 0 -	0000 1001 0 -	0000 0111 0 -
1001 0111	0111 1001	1001 1001	0111 0111
1100 1011 1 x	0011 1100 1 +	1100 1100 1 +	0011 1011 1 x
1110 0101 1 x	1100 1100	0011 1100	0001 1101 1 x
1111 0010 1 +	1110 0110 0 x	0001 1110 0 x	0000 1110 1 +
0110 0010	1111 0011 0 -	0000 1111 0 -	1001 1110
0011 0001	0110 0011	1100 1111	1100 1111
	00110 001		
= 3*16+1		-(0011 0001)	-7*7 = -49
	-7*-7 = 49		

Signed Multiplication

- Convert negative numbers to positive and remember the original signs.
- In 2s-complement, can multiply directly
  - using a 1-bit longer product register
  - Sign extend when shifting
  - Subtract instead of adding on last step
- Or, use Booth's algorithm
  - Sign extend when shifting

Integer Division

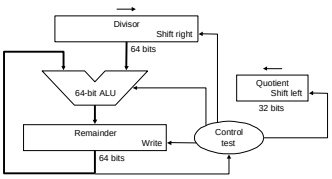
- Dividend = Quotient x Divisor + Remainder
- Example: $1{,}001{,}010_{ten}$ / 1000_{ten}

```
                      1  0  0  1 ten  Quotient
Divisor  1000 ten  | 1  0  0  1  0  1  0 ten  Dividend
                   -1  0  0  0
                      1  0
                      1  0  1
                      1  0  1  0
                     -1  0  0  0
                            1  0 ten  Remainder
```

Compare shifted divisor and remainder, subtract only if divisor<=remainder

Division Hardware #1

- Dividend starts in Remainder register
- Divisor starts in left half of divisor register
- Comparison requires extra hardware, not shown, or extra steps



Division (contd.)

- Similar to multiplication
  - Shift remainder left instead of shifting divisor right
  - Combine quotient register with right half of remainder register
- Signed Division
  - Remember the signs and negate quotient if different.
  - Make sign of remainder match the dividend
- Same hardware can be used for both multiply and divide.
  - Need 64-bit register that can shift left and right
  - ALU that adds or subtracts
  - Optimizations possible

Summary

- Both multiplication and division are MULTISTEP algorithms
- Multiplication takes some 32 “steps”, each about 1 cycle
- Division may take twice as long
- Some optimizations are possible
  - Multiplication in about 9 cycles (32-bit operands)
  - Division in about 32
- Each algorithm uses one 64-bit register, usually split into a HI 32-bit part, and a LO 32-bit part
  - MIPS uses MFHI and MFLO operations to read these
- Both operations (and MOD) are SLOW compared to ADD