NUMERICAL OPERATIONS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Homework 5 is due on Feb 14
  - Verify your uploaded file before the deadline

- This lecture
  - Multiplication/division operations
  - Floating point

Recall: Overflows

- Note: machines use a certain number of bits for representing each number
  - e.g., 32-bit integers

- For an unsigned number, overflow happens when the last carry (1) cannot be accommodated

- For a signed number, overflow happens when the most significant bit is not the same as every bit to its left
  - when the sum of two positive numbers is a negative result
  - when the sum of two negative numbers is a positive result
  - The sum of a positive and negative number will never overflow

MIPS Instructions

- Instructions add, addi, sub may cause exceptions on overflow
  - Software needs to handle exceptions
    - More on this later
- MIPS allows addu and subu instructions that work with unsigned integers and never flag an overflow
  - Other instructions may be executed to detect the overflow

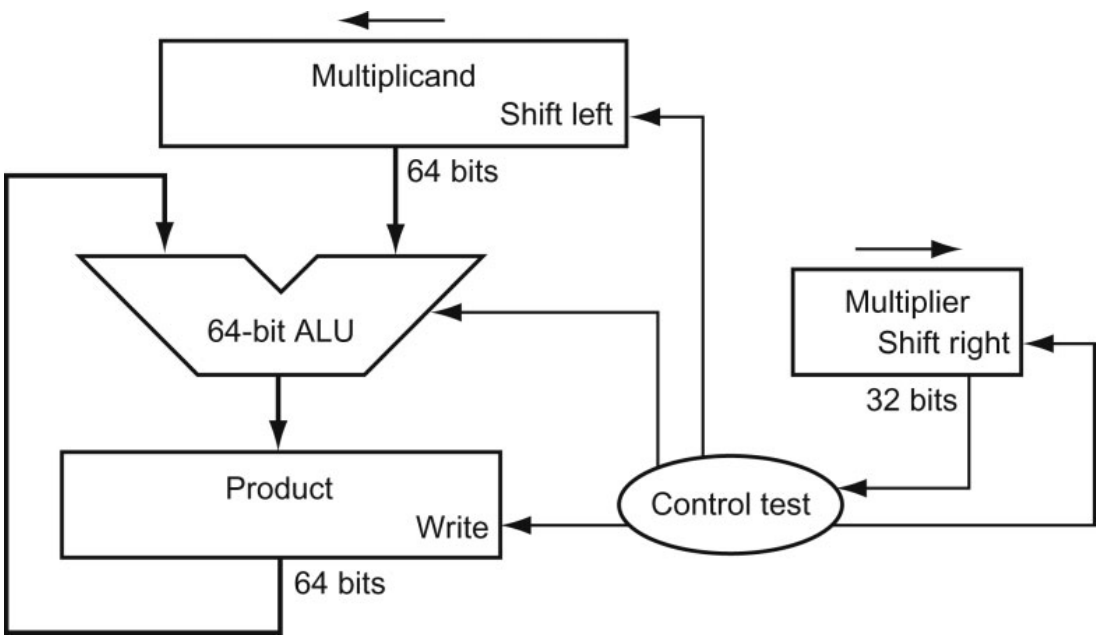
Multiplication Example

- Multi-step process
- Every step
  - multiplicand is shifted
  - next bit of multiplier is examined (also a shifting step)
  - if this bit is 1, shifted multiplicand is added to the product

```
Multiplicand       1000ten
Multiplier     x   1001ten
               ----------------
                   1000
                  0000
                 0000
                1000
               ----------------
Product         1001000ten
```

Multiplication Example

- Multi-step process



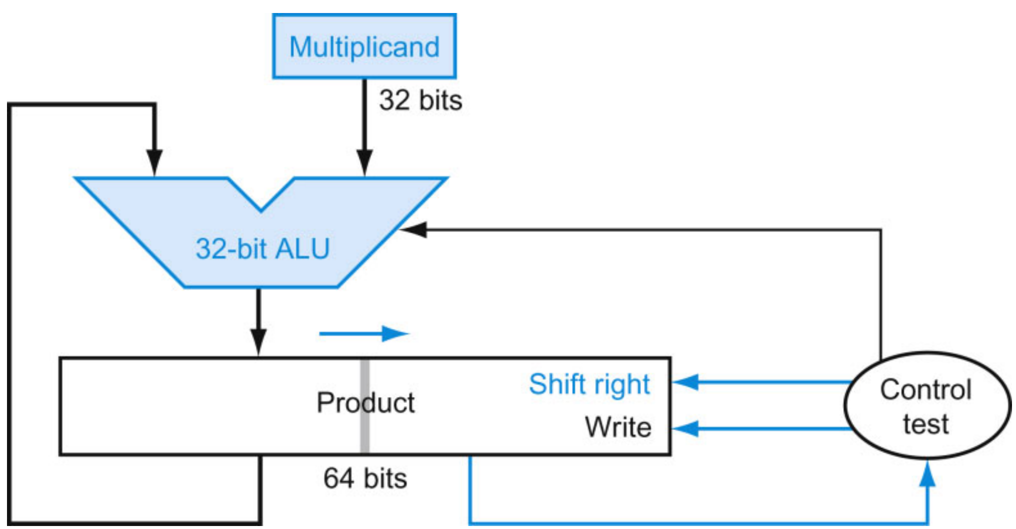
Multiplicand		1000_{ten}
Multiplier	x	1001_{ten}

1000
0000
0000
1000

Product 1001000_{ten}

Multiplication Algorithm 2

- A more efficient algorithm
  - 32-bit ALU and multiplicand is untouched
  - sum keeps shifting right
    - number of bits in product + multiplier = 64,
    - hence, they share a single 64-bit register



Multiplication Notes

- The previous algorithm also works for signed numbers (negative numbers in 2's complement form)
- We can also convert negative numbers to positive, multiply the magnitudes, and convert to negative if signs disagree
- The product of two 32-bit numbers can be a 64-bit number
  - In MIPS, the product is saved in two 32-bit registers

MIPS Instructions

- Signed multiplication (mult)

```
mult    $s2, $s3
```
computes the product and stores it in two "internal" registers that can be referred to as hi and lo

```
mfhi    $s0
mflo    $s1
```
moves the value in hi into $s0
moves the value in lo into $s1

- Similarly for unsigned multiplication (multu)

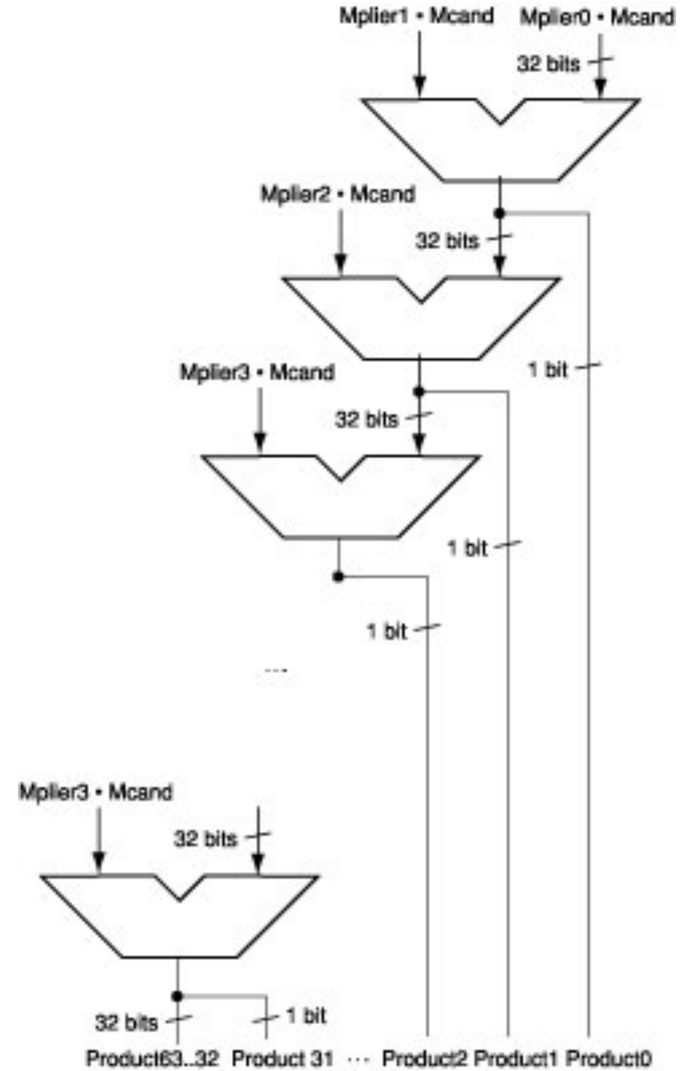
```
multu   $s2, $s3

mfhi    $s0
mflo    $s1
```

Multiplication: Fast Algorithm

- The previous algorithm requires a clock to ensure that the earlier addition has completed before shifting

- This algorithm can quickly set up most inputs – it then has to wait for the result of each add to propagate down – faster because no clock is involved

- Note: high transistor cost



Division Example

- Multi-step process
  - shift divisor right and compare it with current dividend
    - if divisor is larger, shift 0 as the next bit of the quotient
    - if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient

		1001_{ten}	Quotient
Divisor	1000_{ten}	$\vert\ 1001010_{ten}$	Dividend
		$\underline{-1000}$	
		10	
		101	
		1010	
		$\underline{-1000}$	
		10_{ten}	Remainder

Division Example

- Divide 7_{ten} ($0000\ 0111_{two}$) by 2_{ten} (0010_{two})

Iter	Step	Quot	Divisor	Remainder
0	Initial values			
1				
2				
3				
4				
5				

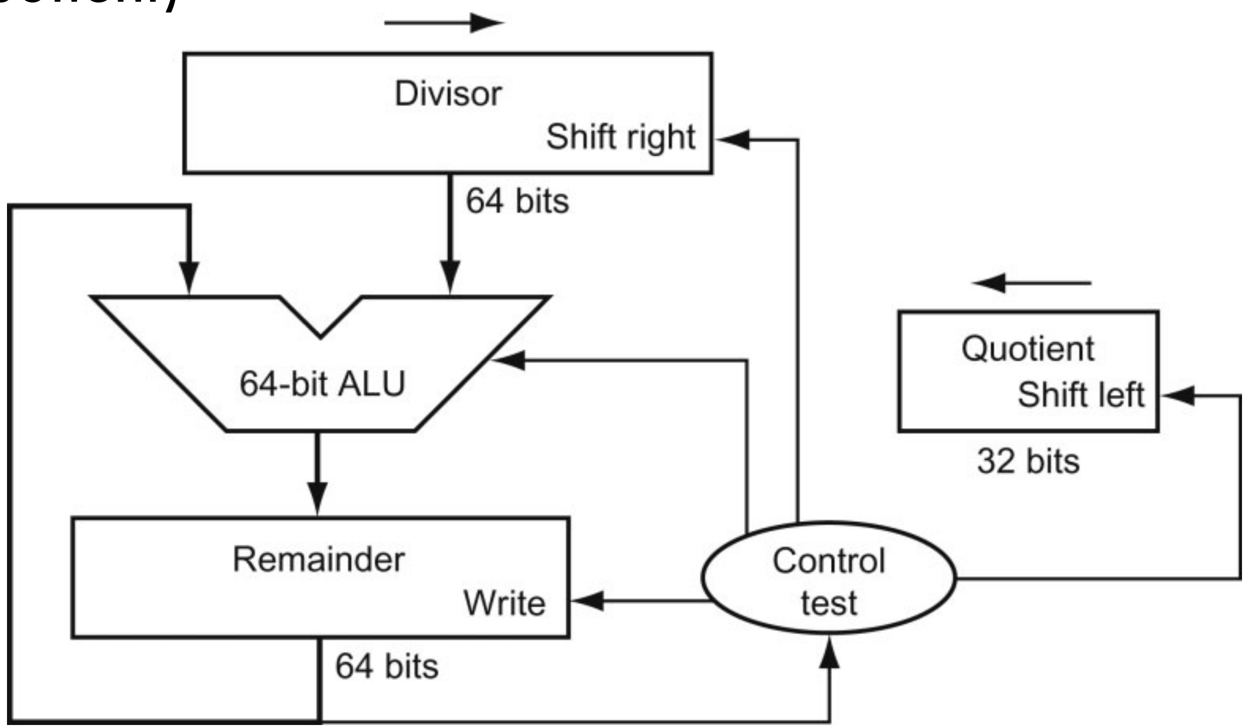
Division Example

- Divide 7_{ten} ($0000\ 0111_{two}$) by 2_{ten} (0010_{two})

Iter	Step	Quot	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	Rem = Rem – Div	0000	0010 0000	1110 0111
	Rem < 0 ➔ +Div, shift 0 into Q	0000	0010 0000	0000 0111
	Shift Div right	0000	0001 0000	0000 0111
2	Same steps as 1	0000	0001 0000	1111 0111
		0000	0001 0000	0000 0111
		0000	0000 1000	0000 0111
3	Same steps as 1	0000	0000 0100	0000 0111
4	Rem = Rem – Div	0000	0000 0100	0000 0011
	Rem >= 0 ➔ shift 1 into Q	0001	0000 0100	0000 0011
	Shift Div right	0001	0000 0010	0000 0011
5	Same steps as 4	0011	0000 0001	0000 0001

Hardware for Division

- A comparison requires a subtract; the sign of the result is examined; if the result is negative, the divisor must be added back
- Similar to multiply, results are placed in Hi (remainder) and Lo (quotient)



Efficient Hardware for Division

- A comparison requires a subtract; the sign of the result is examined; if the result is negative, the divisor must be added back
- Similar to multiply, results are placed in Hi (remainder) and Lo (quotient)