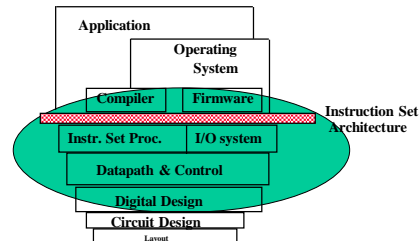
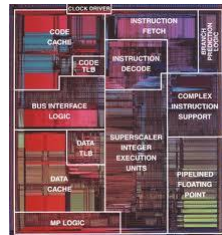


CS/SE 3340

Computer Architecture

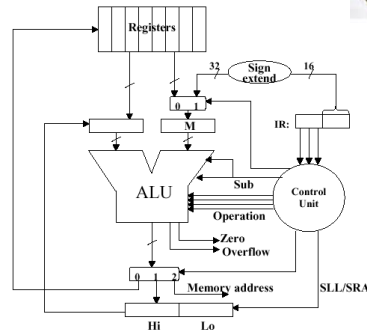
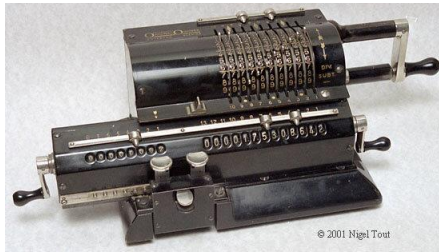


Integer Arithmetic

Adapted from "Computer Organization and Design, 4th Ed." by D. Patterson and J. Hennessy

Questions

- How to do addition/subtraction in binary?
- What is 'overflow', Why/When does it happen and How do computers deal with it ?
- How to build a single bit/multibit binary adder?
- What is a ripple-carry binary adder, why is it slow and how to build a fast adder?
- How to do binary multiply/divide in hardware?
- What are MIPS instructions for multiply/divide?

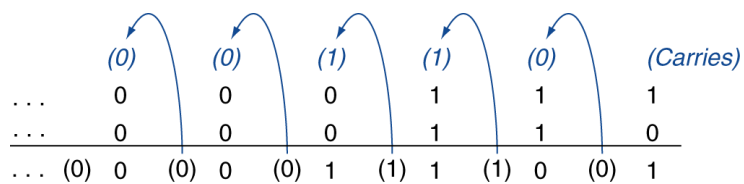


<http://vintagecalculators.com>

3

Integer Addition

• Example: 7 + 6



- Overflow if result out of range
 - Adding **+ui** and **-ui** operands, no overflow
 - Adding two **+ui** operands
 - Overflow if result MSbit is **1**
 - Adding two **-ui** operands
 - Overflow if result MSbit is **0**

4

Integer Subtraction

- Add negation of second operand

- Example: $7 - 6 = 7 + (-6)$

```
+7:  0000 0000 ... 0000 0111
-6:  1111 1111 ... 1111 1010
-----
+1:  0000 0000 ... 0000 0001
```

- Overflow if result out of range
 - Subtracting two **+ui** or two **-ui** operands, no overflow
 - Subtracting **+ui** from **-ui** operand
 - Overflow if result MSbit is 0
 - Subtracting **-ui** from **+ui** operand
 - Overflow if result MSbit is 1

5

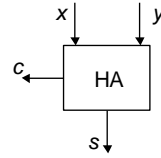
Dealing with Overflow

- 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

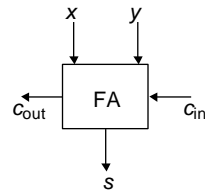
6

Binary Adders

Inputs		Outputs	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Inputs			Outputs	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

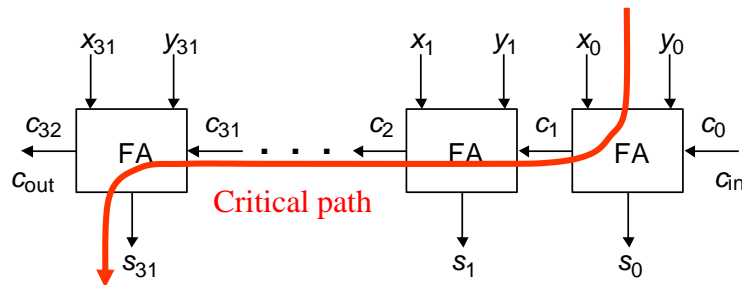


These adders can be arranged in parallel to form n-bit adders

Behrooz Parhami, "Computer Architecture"

7

Ripple Carry Adder



Simple but slow – why?

Behrooz Parhami, "Computer Architecture"

8

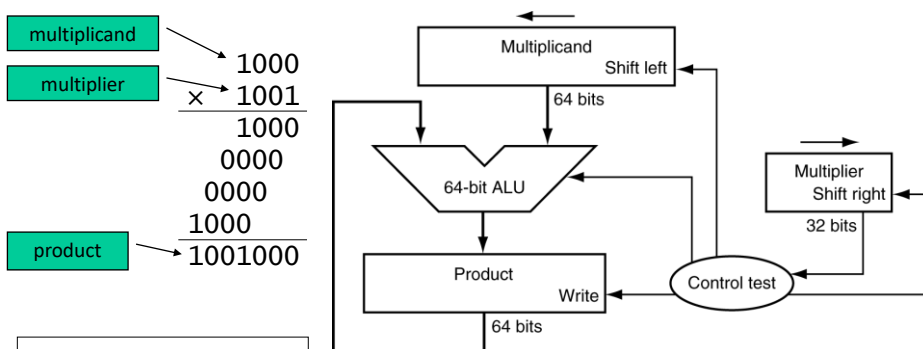
Arithmetic for Multimedia

- 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

9

Multiplication

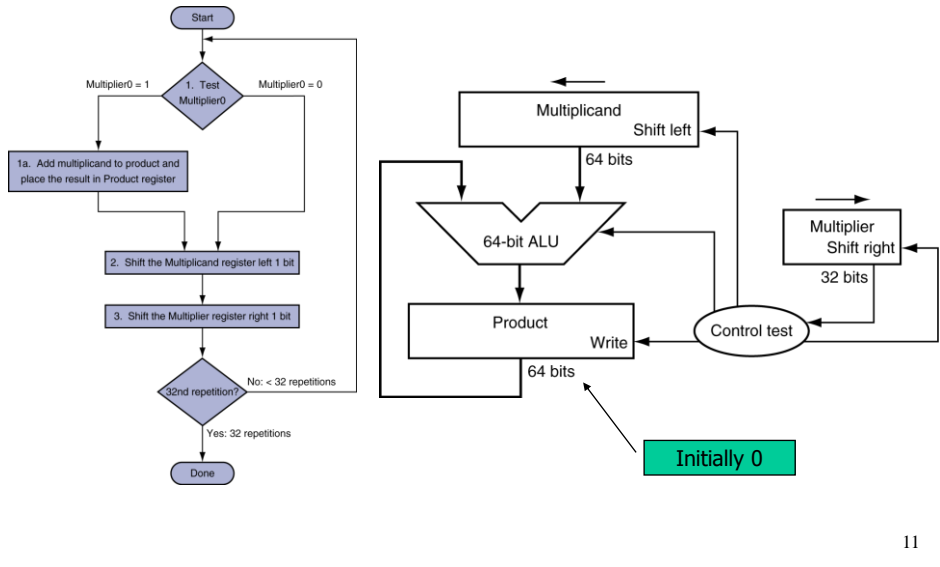
- Start with long-multiplication approach



Length of product is the sum of operand lengths

10

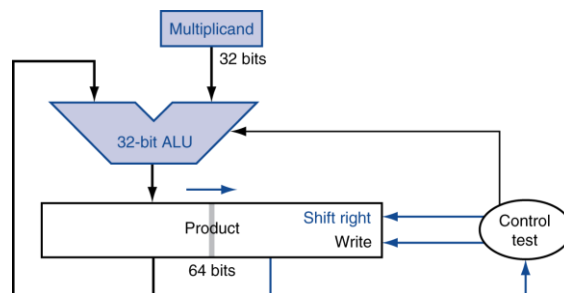
Multiplication Hardware



11

Optimized Multiplier

- Perform *steps in parallel: add/shift*

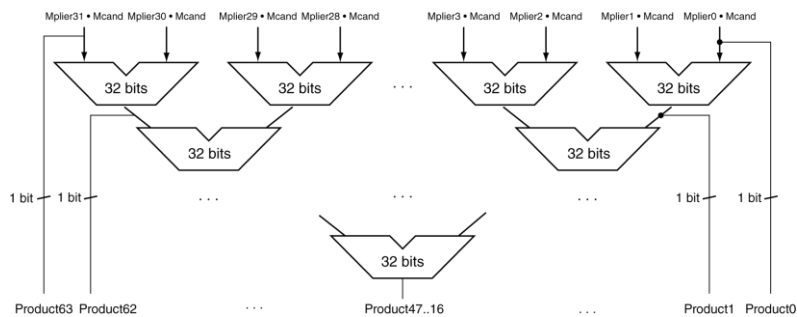


- One cycle per partial-product addition
 - That's ok, if frequency of multiplications is low

12

Faster Multiplier

- Uses multiple adders
 - Cost/performance tradeoff



- Can be pipelined
 - Several multiplication performed in parallel

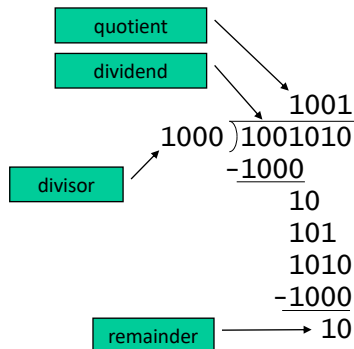
13

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`

14

Division

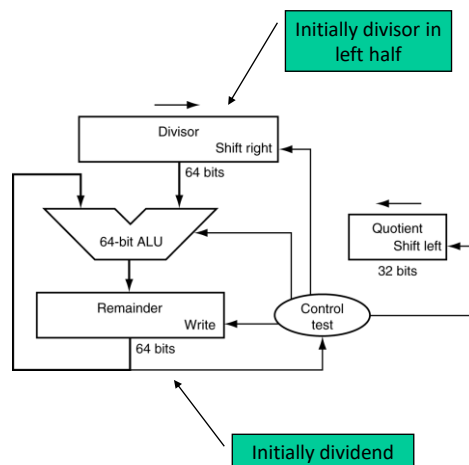
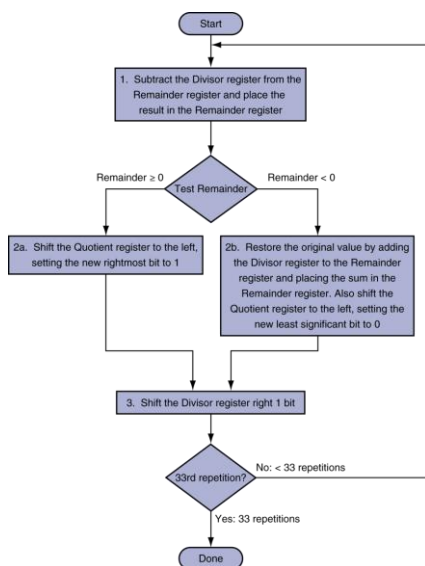


n -bit operands yield n -bit quotient and remainder

- Check for 0 divisor
- Long division approach
 - If divisor \leq 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

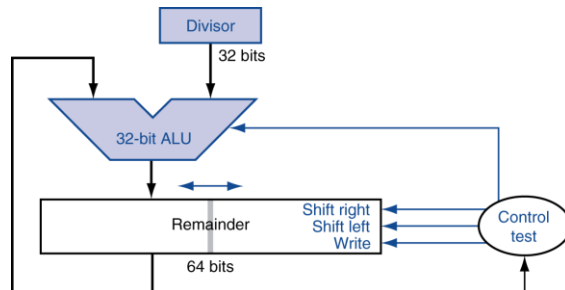
15

Division Hardware



16

Optimized Divider



- One cycle per partial-remainder subtraction
- Looks a lot like a multiplier!
 - Same hardware can be used for both

17

Faster Division

- Can't use parallel hardware as in multiplier
 - Subtraction is conditional on sign of remainder
- Faster dividers (e.g. SRT division) generate multiple quotient bits per step
 - Still require multiple steps

18

MIPS Division

- Use HI/LO registers for result
 - HI: 32-bit remainder
 - LO: 32-bit quotient
- Instructions
 - `div rs, rt` / `divu rs, rt`
 - No overflow or divide-by-0 checking
 - Software must perform checks if required
 - Use `mfhi`, `mflo` to access result