
IC220
Set #10:
More Computer Arithmetic (Chapter 3)

1

ADMIN

- Read pages 259-262 (MIPS floating point instructions)
- Read 3.8

2

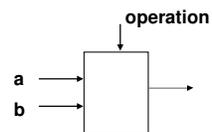
An Arithmetic Logic Unit (ALU)

The ALU is the 'brawn' of the computer

- What does it do?

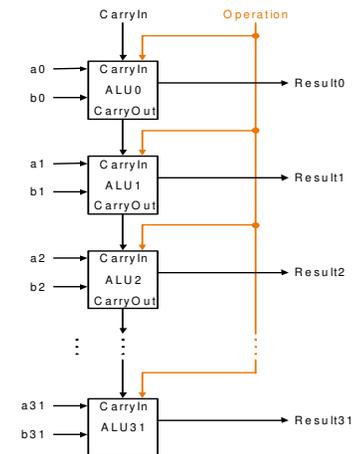
- How wide does it need to be?

- What outputs do we need for MIPS?



3

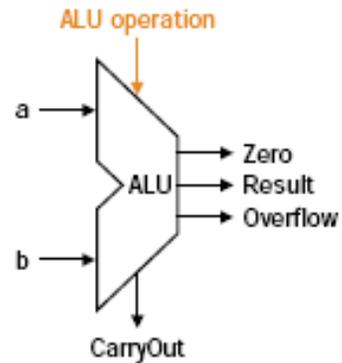
A simple 32-bit ALU



4

ALU Control and Symbol

ALU Control Lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract
0111	Set on less than
1100	NOR



5

Multiplication

- More complicated than addition
 - accomplished via shifting and addition
- Example: grade-school algorithm

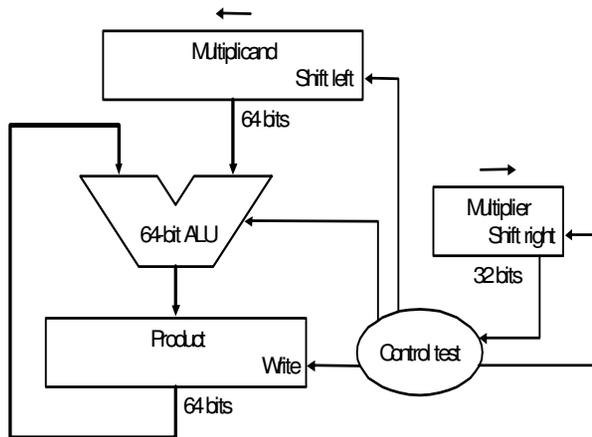
```

      0010 (multiplicand)
    x 1011 (multiplier)
    -----
    
```

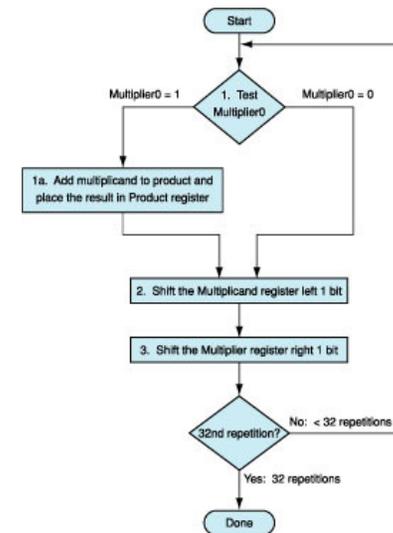
- Multiply $m * n$ bits, How wide (in bits) should the product be?

6

Multiplication: Simple Implementation



7



Using Multiplication

- Product requires 64 bits
 - Use dedicated registers
 - HI – more significant part of product
 - LO – less significant part of product
- MIPS instructions


```
mult $s2, $s3
multu $s2, $s3
mfhi $t0
mflo $t1
```
- Division
 - Can perform with same hardware! (see book)

```
div $s2, $s3      Lo = $s2 / $s3
                  Hi = $s2 mod $s3

divu $s2, $s3
```

9

Floating Point

- We need a way to represent
 - numbers with fractions, e.g., 3.1416
 - very small numbers, e.g., .00000001
 - very large numbers, e.g., 3.15576×10^{23}
- Representation:
 - sign, exponent, significand:
 - $(-1)^{\text{sign}} \times \text{significand} \times 2^{\text{exponent (some power)}}$
 - Significand always in normalized form:
 - Yes:
 - No:
 - more bits for significand gives more
 - more bits for exponent increases



10

IEEE754 Standard

Single Precision (float): 8 bit exponent, 23 bit significand

31	30	29	28	27	26	25	24	23	22	21	20	.	.	.	9	8	7	6	5	4	3	2	1	0
S Exponent (8 Bits)									Significand (23 bits)															

Double Precision (double): 11 bit exponent, 52 bit significand

31	30	29	28	.	.	.	21	20	19	18	17	.	.	.	9	8	7	6	5	4	3	2	1	0
S Exponent (11 Bits)											Significand (20 bits)													
More Significand (32 more bits)																								

11

IEEE 754 – Optimizations

- Significand
 - What's the first bit?
 - So...
- Exponent is “biased” to make sorting easier
 - Smallest exponent represented by:
 - Largest exponent represented by:
 - Bias values
 - 127 for single precision
 - 1023 for double precision
- Summary: $(-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent} - \text{bias}}$

12

Example:

- Represent -9.75_{10} in binary, single precision form:
- Strategy
 - Transfer into binary notation (fraction)
 - Normalize significand (if necessary)
 - Compute exponent
 - (Real exponent) = (Stored exponent) - bias
 - Apply results to formula
 - $(-1)^{\text{sign}} \times (1 + \text{significand}) \times 2^{\text{exponent} - \text{bias}}$

13

Floating Point Complexities

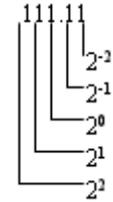
- Operations are somewhat more complicated (see text)
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
 - IEEE 754 keeps two extra bits, guard and round
 - four rounding modes
 - positive divided by zero yields “infinity”
 - zero divide by zero yields “not a number”
 - other complexities
- Implementing the standard can be tricky

15

Example continued:

Represent -9.75_{10} in binary single precision:

- $-9.75_{10} =$



- Compute the exponent:
 - Remember ($2^{\text{exponent} - \text{bias}}$)
 - Bias = 127

- Formula $(-1)^{\text{sign}} \times (1 + \text{significand}) \times 2^{\text{exponent} - \text{bias}}$

31	30	29	28	27	26	25	24	23	22	21	20	.	.	9	8	7	6	5	4	3	2	1	0	

MIPS Floating Point Basics

- Floating point registers
 - $\$f0, \$f1, \$f2, \dots, \$f31$
 - Used in pairs for double precision ($f0, f1$) ($f2, f3$), ...
 - $\$f0$ not always zero
- Register conventions:
 - Function arguments passed in
 - Function return value stored in
 - Where are addresses (e.g. for arrays) passed?
- Load and store:
 - `lwc1 $f2, 0($sp)`
 - `swc1 $f4, 4($t2)`

16

MIPS FP Arithmetic

- Addition, subtraction: `add.s`, `add.d`, `sub.s`, `sub.d`
`add.s $f1, $f2, $f3`
`add.d $f2, $f4, $f6`
- Multiplication, division: `mul.s`, `mul.d`, `div.s`, `div.d`
`mul.s $f2, $f3, $f4`
`div.s $f2, $f4, $f6`

17

Example #1

- Convert the following C code to MIPS:

```
float max (float A, float B) {  
    if (A <= B) return A;  
    else      return B;  
}
```

19

MIPS FP Control Flow

- Pattern of a comparison: `c.____.s` (or `c.____.d`)
`c.lt.s $f2, $f3`
`c.ge.d $f4, $f6`
- Where does the result go?
- Branching:
`bc1t label10`
`bc1f label20`

18

Example #2

EX: 3-21 ...

- Convert the following C code to MIPS:

```
void setArray (float F[], int index,  
              float val) {  
    F[index] = val;  
}
```

20

Chapter Three Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
 - two's complement
 - IEEE 754 floating point
- Computer instructions determine “meaning” of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines (i.e., algorithms and implementation).

- We are (almost!) ready to move on (and implement the processor)

Chapter Goals

- Introduce 2's complement numbers
 - Addition and subtraction
 - Sketch multiplication, division
- Overview of ALU (arithmetic logic unit)
- Floating point numbers
 - Representation
 - Arithmetic operations
 - MIPS instructions