**Computer**

**Architecture** CH3 Computer Arithmetic (I)

Prof. Ren-Shuo Liu 

NTHU EE

Fall 2017

CH2

CH3, 4, 5

Applications

Data Structures / Algorithms **Programming**

Instruction Set Archtecture Organization & Architecture **Logic Design**

**Digital Electronics**

Solid-State Electronics



Software

Hardware

Computer

Architecture

2

Outline • Overview

• Integer operations

• Bit-wise logical operations

• Additions, subtractions

• Comparisons

• Multiplications

• Divisions

• Floating point operations

• Additions, subtractions

• Multiplications

3

Outline • **Overview**

• Arithmetic logic unit (ALU)

• Inter instructions

• Logic gates

• Value representation and interpretation

• Integer operations

• Floating point operations

4

Common Integer Instructions • Bit-wise logical

• and, andi

• or, ori

• xor, xori

• nor

• Arithmetic

• add, addi

• sub

• mult

• div

• Comparisons

• slt, slti, sltu, sltiu

5

Arithmetic Logic Unit (ALU) ADDI R1, R2, 100

… …

Immediate

R0

R1

R2

R3

…

…

R31

… …

Operation

**ALU**

(integer)

Zero

Overflow Results

Integer ALU Block Overview 

7

Logic Gates 

**In** … and **Out** … or not

**In Out** all 1's 1 others 0

**In Out** all 0's 0 others 1

**In Out** 0 1 1 0

8

Logic Gates (Cont'd) 

**In0**

**Sel**

0

**Out** … … xor… ...

nor

**In1 In*k***

1 *k*

mux

**In Out** all 0's 1 others 0

**In Out**

odd num. of 1's 1 others 0

**Sel Out** 0 In0 1 In1 … …

k In*k*

9

Value Representation and Interpretation 

Representation (表示方式) 11 1 0 1 1 Interpretation (解讀方式)11 1 0 1 1

10

Value Representation and Interpretation 

Two representation strategies (兩種表示方式) 1 0 1 1

11

0 0 0 0

(Excess-11)

11

Value Representation and Interpretation 

Two interpretation strategies (兩種解讀方式) 11

1 0 1 1

(sing-magnitude) -3

12

Binary Unsigned Integers a 32-bit register

1 1 … 0 1 0 0 0 1 0 0 1 0 1 0 0 1 …

31 30 … 12 … 8 5 4 3 2 1 0

• Above is interpreted as (231+230+212+28+25+23+20) • Range

• Min: 0

• Max: 233-1

13

Unsigned Integers (Cont'd) 

0000….0000 (Min)

1111….1111 (Max) 0000….0001 1111….1110 0000….0010

N-bit unsigned integers

can have 2N values

14

Unsigned Integers (Cont'd) 

0000….0000 (Min)

1111….1111 (Max) 0000….0001 1111….1110 0000….0010

15

Signed Integers • Many widely-used representations

• Biased/offset/excess-K

• Signed magnitude

• Two's complement

16

Biased / Offset / Excess-K a 32-bit register

0 1 … 0 1 0 0 0 1 0 0 1 0 1 0 0 1 …

31 30 … 12 … 8 5 3 0 • Above is interpreted as

• (230+212+28+25+23+20) - K

• Where K is a predefined value

• Range

• Most negative: -K

• Most positive: 232- K - 1

17

Sign Magnitude Representation a 32-bit register

S 1 … 0 1 0 0 0 1 0 0 1 0 1 0 0 1 …

31 30 … 12 … 8 5 3 0 • Above is interpreted as

• S × (230+212+28+25+23+20)

• (230+212+28+25+23+20) if sign == 0 (i.e., positive) • - (231- (230+212+28+25+23+20) ) if sign == 1 (i.e., negative) • Range

• Most negative: -231

• Most positive: 231- 1

18

2's Compliment Signed Integers a 32-bit register

S 1 … 0 1 0 0 0 1 0 0 1 0 1 0 0 1 …

31 30 … 12 … 8 5 4 3 2 1 0 • Above is interpreted as

• S×(-231) + (230+212+28+25+23+20)

• (230+212+28+25+23+20) if sign == 0 (i.e., positive) • - (231- (230+212+28+25+23+20) ) if sign == 1 (i.e., negative) • Range

• Most negative: -(231 – 1)

• Most positive: 231- 1

19

2's Compliment Signed Integers (Cont'd)

1111….1111 (i.e., -1ten) 1111….1110 (i.e., -2ten) 1111….1101 (i.e., -3ten)

1000….0001

(i.e., -(231-1)ten)

1000….0000

(i.e., -231ten)

(Most negative)

Negative

0000….0000 (i.e., 0ten) 0000….0001 (i.e., 1ten) 0000….0010 (i.e., 2ten)

0111….1110

(i.e., (231-2)ten)

0111….1111

(i.e., (231-1)ten)

(Most positive)

2's Compliment Signed Integers (Cont'd)

1111….1111 (i.e., -1ten) 1111….1110 (i.e., -2ten) 1111….1101 (i.e., -3ten)

1000….0001

(i.e., -(231-1)ten)

1000….0000

(i.e., -231ten)

(Most negative)

0000….0000 (i.e., 0ten) 0000….0001 (i.e., 1ten) 0000….0010 (i.e., 2ten)

0111….1110

(i.e., (231-2)ten)

0111….1111

(i.e., (231-1)ten)

(Most positive)

2's Compliment Signed Integers (Cont'd)

1111….1111 (i.e., -1ten) 1111….1110 (i.e., -2ten) 1111….1101 (i.e., -3ten)

1000….0001

(i.e., -(231-1)ten)

1000….0000

(i.e., -231ten)

(Most negative)

B vs ~~B~~

0000….0000 (i.e., 0ten) 0000….0001 (i.e., 1ten) 0000….0010 (i.e., 2ten)

0111….1110

(i.e., (231-2)ten)

0111….1111

(i.e., (231-1)ten)

(Most positive)

2's Compliment Signed Integers (Cont'd)

1111….1111 (i.e., **-1ten**) 1111….1110 (i.e., **-2ten**) 1111….1101 (i.e., **-3ten**)

0000….0000 (i.e., **0ten**) 0000….0001 (i.e., **1ten**) 0000….0010 (i.e., **2ten**)

-B = ~~B~~ + 1 = 2N-B~~B~~ = -(B + 1) = 2N-(B + 1)

1000….0001 (i.e., **-(231-1)ten**)

1000….0000 (i.e., **-231ten**) (Most negative)

0111….1110 (i.e., **(231-2)ten**)

0111….1111 (i.e., **(231-1)ten**) (Most positive)

Outline • Overview

• Integer operations

• Bit-wise logical operations

• Additions, subtractions

• Comparisons

• Multiplications

• Divisions

• Floating point operations

• Additions, subtractions

• Multiplications

24

Bit-Wise Logical Operations • Common CPU instructions

• and, andi, or, ori, nor, xor, xori

• Example

• and R1, R4, R5 // R1 = (R4 & R5)

1 1 … 0 1 0 0 0 1 0 0 1 0 1 0 0 1 R4

R5

0 0 … 0 0 0 0 0 0 1 1 1 1 1 1 1 1

R1 0 0 … 0 0 0 0 0 0 0 0 1 0 1 0 0 1

25

Supporting AND is Easy 

A[31:0] B[31:0]

A[0] B[0]

A[1] B[1]

A[2] B[2]

A[31] B[31]

… …

S[0]

S[1]

S[2]

S[31]

S[31:0]

26

Support AND (Block Diagram) 

A[31:0] B[31:0]

A[0] B[0]

A[1] B[1]

A[2] B[2]

A[31] B[31]

… …

S[0]

S[1]

S[2]

S[31]

A[i]

B[i]S[i]S[31:0]

27

Support OR 

A[31:0] B[31:0]

A[0] B[0]

A[1] B[1]

A[2] B[2]

A[31] B[31]

… …

S[0]

S[1]

S[2]

S[31]

A[i]

B[i]S[i]S[31:0]

28

Support Both AND and OR 

Op Op

A[31:0]

A[0]

B[0]

A[1]

B[1]

A[2]

B[2]

A[31]

B[31]

B[31:0]

… …

S[0]

S[1]

S[2]

S[31]

A[i]

B[i]S[i]S[31:0]

29

Support All Logical Instructions • and, andi, or, ori, xor, xori, nor

Op

A[31:0]

A[0]

B[0]

A[i]

B[i]

S[0]

S[i]

Op

and

andi

or

ori

B[31:0]

A[1] B[1]

A[2] B[2]

A[31] B[31]

… …

S[1] S[2]

S[31]

S[31:0]

xor xori

nor

30

Further Support ADD

Carry in

(Cin[i])

~~L~~og~~ic~~

A[i]

Op

and

or …

Op

A[31:0]

A[0]

B[0]

A[1]

B[i]

S[0]

S[i]

S[1]

op~~.~~

xor … nor

B[1]

A[2]

B[2]

A[31]

B[31]

B[31:0]

… …

S[2] S[31]

S[31:0]

+

(FA)

Carry out (Cout[i])

**add**

**addi**

31

1-Bit Full Adder (FA) • Outputs = the sum of

cin[i] A[i]

B[i]

+

(FA)

S[i] Cout[i] 

the three input bits

**Cin[0] A[i] B[i] Cout[i] S[i]** 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

32

1-Bit Full Adder (FA) • Some observatios

cin[i] A[i]

B[i]

+

(FA)

S[i] Cout[i] 

• Three inputs are interchangeable

• Outputs = the

number of 1's in the inputs

• S is 1 if the number of 1's is odd (i.e., XOR) • Cout (進位) is 1 if the number of 1's ≥ 2

**Cin[0] A[i] B[i] Cout[i] S[i]** 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

33

1-Bit Half Adder (HA)

• Outputs = the sum of

A[i] B[i]

~~+~~

~~(H~~A~~)~~

~~~~S[i] Cout[i] 

the two input bits

**A[i] B[i] Cout[i] S[i]** 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0

34

1-Bit Half Adder (HA)

~~+~~

A[i]

S[i]

• Some observations

B[i]

~~(H~~A~~)~~

Cout[i]

• Two inputs are

interchangeable

• Outputs = the

number of 1's in the inputs

• S is 1 if the number of 1's is odd (i.e., XOR) • Cout (進位) is 1 if both the inputs are 1's (i.e., AND)

**A[i] B[i] Cout[i] S[i]** 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0

35

Building a 1-Bit Full Adder 

Cin[i] S[i] +

A[i]

(HA)

~~+~~

One of A and B is 1

Cout[i]

B[i]

~~(HA)~~

Both A and B are 1's 🡪 a carry

36

Building a 1-Bit Full Adder (Cont'd)

Cin[i] S[i]

One of

A and B

A[i] B[i]

is 1

Both A and B are 1's 🡪 a carry

Cout[i]

37

Addition A[31:0]

Cin[0]



Op Op Cin[i]

A[0]

S[0]

A[i]

logi.

B[0]

A[1] B[1]

S[1]

B[i]S[i] +

A[2] B[2]

A[31] B[31]

… …

S[2]

S[31]

S[31:0]

Cout[i]

B[31:0]

Cout[31]

38

Subtraction • Subtraction can be done using

an addition and an negation(取負)

0

• A - B

= A + (-B)

= A + (~~B~~ + 1)

• E.g., in 5-bit (modulo 32) arithmetic

=

• **7** - **11**

= 7 + (-11)

0

= 7 + 21 // 32-11 = 21; (11+1) = 21

= -4 // interpreted as singed

39

Subtraction 

• B negation

(i.e, -B = ~~B~~ + 1)

• Inversion (0-1互換) can

Bneg

Carry in

~~L~~og~~ic~~

Op

be done using XOR or mutiplexer

A[i]

B[i]

S[i]

op~~.~~

B[i]

Bneg 0

1

+

Carry out

40

Subtraction

• B negation

(i.e, -B = B + 1)

• +1 is done by using Cin[0]

BNeg

Cin[0]

A[0]

B[0]

A[1]

B[1]

A[2]

B[2]

A[31]

B[31]

Op … …

S[0]

S[1]

S[2]

S[31]

Cout[31]

41

Zero DetectionBNeg

Op 

• Indicate whether the output 32-bit result is a zero

Cin[0]

A[0]

B[0]

A[1]

B[1]

A[2]

B[2]

A[31]

B[31]

… …

S[0]

S[1]

S[2]

S[31]

zero

…

Cout[31]

42

Overflow Detection • Before describing overflow detection methods, some things need to be clarified

• Overflow detection methods are different for signed and unsigned operations

• Comparisons (SLT and SLTU) are related to overflow detection

43

Overflow Detection (Signed Add)

| **A**  | **B**  | **S=A+B**  | **overflow?** |
| --- | --- | --- | --- |
| **+**  | **+**  | **-**  | Yes |
| -  | -  | +  | Yes |
| **+**  | **+**  | **+**  | No |
| -  | -  | -  | No |
| +  | - | (overflow cannot occur) |
| -  | + |
| 0  | + |
| 0  | - |
| +  | 0 |
| -  | 0 |

44

Overflow Detection (Signed Add) • According to the above descriptions

| **A[31]**  | **B[31]**  | **Cin[31]**  |  | **Cout[31]**  | **S[31]**  |  | **Overflow** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0  | 0  | 0  |  | 0  | 0  |  | 0 |
| 0  | 0  | 1  |  | 0  | 1  |  | **Yes** |
| 0  | 1  | 0  |  | 0  | 1  |  | 0 |
| 0  | 1  | 1  |  | 1  | 0  |  | 0 |
| 1  | 0  | 0  |  | 0  | 1  |  | 0 |
| 1  | 0  | 1  |  | 1  | 0  |  | 0 |
| 1  | 1  | 0  |  | 1  | 0  |  | **Yes** |
| 1  | 1  | 1  |  | 1  | 1  |  | 0 |

45

Overflow Detection (Signed Add) • Tricky approach

| **A[31]**  | **B[31]**  | **Cin[31]**  |  | **Cout[31]**  | **S[31]**  |  | **Overflow** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0  | 0  | 0  |  | 0  | 0  |  | 0 |
| 0  | 0  | 1  |  | 0  | 1  |  | **Yes** |
| 0  | 1  | 0  |  | 0  | 1  |  | 0 |
| 0  | 1  | 1  |  | 1  | 0  |  | 0 |
| 1  | 0  | 0  |  | 0  | 1  |  | 0 |
| 1  | 0  | 1  |  | 1  | 0  |  | 0 |
| 1  | 1  | 0  |  | 1  | 0  |  | **Yes** |
| 1  | 1  | 1  |  | 1  | 1  |  | 0 |

46

Overflow Detection (Unsigned Add) 

| **A**  | **B**  | **A + B overflow?** |
| --- | --- | --- |
| non-zero  | non-zero  | If (A+B) produces a carry |
| 0  | non-zero | (cannot occur) |
| non-zero  | 0  |
| 0  | non-zero |

Overflow condition: If (A+B) produces a carry

47

Overflow Detection (Unsigned Sub) 

| **A**  | **B**  | **A - B = A + (-B) overflow?** |
| --- | --- | --- |
| non-zero  | non-zero  | If (A-B) > A (i.e., if A + (-B) does not produce a carry)  |
| 0  | non -zero |
| non-zero  | 0 | (cannot occur) |
| 0  | 0 |

Overflow condition: If A + (-B) does not produce a carry

48

Overflow Detection (Unsigned Operands) • 7 + 11

0

0 0 1 1 1

+ 0 1 0 1 1

0 1 0 0 1 0

• 7 + 26 (overflow)

0 0 1 1 1

0

+ 1 1 0 1 0

1 0 0 0 0 1

carry==1 during unsigned add

indicates overflow

49

Overflow Detection (Unsigned Operands) • 7 - 11

= 7 + (-11)

0

0

= 7 + 21

= 28 ☹ carry==0 during unsigned sub indicates overflow

=

0 0 1 1 1

+ 1 0 1 0 1

0

0 1 1 1 0 0

50

Overflow Detection

Op

BNeg

Cin[0]

A[0]

S[0]

Op

A[i]

Bneg

Carry in

Logic

B[0]

A[1] B[1]

zero

…

S[1]

B[i]

S[i]

op.

A[2] B[2]

S[2]

+

overflow

A[31] B[31]

… …

S[31]

overflow

Carry out

Cout[31]

51

More Discussions on Overflow • Different languages have different designs

| **Language**  | **Unsigned integer**  | **Signed integer** |
| --- | --- | --- |
| **Ada**  | modulo the type’s modulus  | raise Constraint\_Error |
| **C/C++**  | modulo power of two  | undefined behavior |
| **C#**  | modulo power of 2 in unchecked context; System.OverflowException is raised in checked context |
| **Java**  | NA  | ignored |
| **Python 2**  | NA  | convert to long |
| **Seed7**  | NA  | raise OVERFLOW\_ERROR |
| **Scheme**  | NA  | convert to bigNum |
| **Smalltalk**  | NA  | convert to LargeInteger |
| **Swift**  | Causes error unless using special overflow operators. |

wikipedia

52

SLT and SLTU (Set Less Than) • Requirements

• ALU outputs 1 (i.e., 0000….0001) if A < B, otherwise outputs 0 • Methods

• Leverage SUB and SUBU hardware

| **SLT** | **Operation** S=A-B= A+~~B+~~1 | **Overflow** 0 | **Sign(S)** **(i.e., S[31])** 1 | **Carry out** **(i.e., S[32])** Don’t Care | **Outcome** 1 (A<B) |
| --- | --- | --- | --- | --- | --- |
| 0  | 0 (A≧B) |
| 1 | 1  | 0 (A≧B) |
| 0  | 1 (A<B) |
| **SLTU**  |
| Don’t care | 1  | 0 (A≧B) |
| 0  | 1 (A<B)53  |

SLT (Set Less Than)

Op 

BNeg

Op

A[i]

Bneg

Carry in

Logic

Cin[0]

A[0]

B[0]

S[0]

zero

B[i]

S[i]

op.

0

…

S[1]

For the 31th

+

slt

overflow

0 0

… …

S[2]

S[31]

overflow

bit only

set slt

set

sltu

54

Summary • Bit-wise logical 

• and, andi 

• or, ori 

• xor, xori 

• nor

• Arithmetic 

• add, addi 

• sub, subu

• mul

• div

• Comparisons

• slt, slti, sltu, sltiu

55

Outline • Overview

• Integer operations

• Bit-wise logical operations

• Additions, subtractions

• Comparisons

• Multiplications

• Divisions

• Floating point operations

• Additions, subtractions

• Multiplications

56