

EEC 483 Computer Organization (Spring 2006)

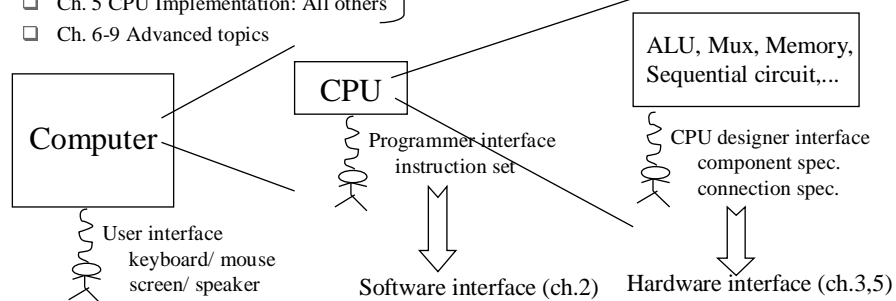
Chapter 3. Arithmetic for Computers

Chansu Yu

Cleveland State University

Table of Contents

- Ch.1, 4 Introduction & Performance
 - Ch. 2 Instruction: Machine Language
 - Ch. 3 CPU Implementation: Arithmetic
 - Ch. 5 CPU Implementation: All others
 - Ch. 6-9 Advanced topics
- Key parts of this course



Cleveland State
University

c.yu91@csuohio.edu

Table of Contents

- Ch. 1, 4 Introduction & Performance
- Ch. 2 Instruction: Machine Language
 - 2.1 Introduction
 - 2.2 Operations (arithmetic, memory operations)
 - 2.3 Operands
 - 2.4 Representing instructions
 - 2.5 Logical operations
 - 2.6 Control flow operations
 - 2.7 Supporting procedures
 - 2.8 Beyond numbers
 - 2.9 MIPS addressing
 - 2.10 Starting a program
- Ch. 3 CPU Implementation: Arithmetic
 - 3.1 Introduction
 - 3.2 Signed and unsigned numbers
 - 3.3 Addition and subtraction
 - 3.4 Multiplication
 - 3.5 Division
 - 3.6 Floating point
 - Appendix B Constructing an Arithmetic Logic Unit (ALU)
- Ch. 5 CPU Implementation: All others
- Ch. 6-9 Advanced topics

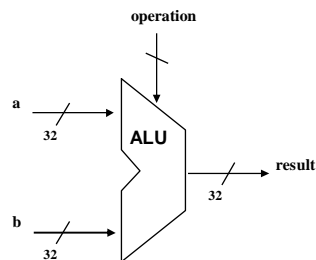
Software interface

**Key parts
of this course**

Hardware interface

Arithmetic

- Where we've been:
 - Performance (seconds, cycles, instructions)
 - Abstractions:
 - Instruction Set Architecture
 - Assembly Language and Machine Language
- What's up ahead:
 - Implementing the Architecture



3.2 Signed and Unsigned Numbers

- ❑ Bits are just bits (no inherent meaning)
 - conventions define relationship between bits and numbers

- ❑ Binary numbers (base 2)
 - 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
 - decimal: $0 \dots 2^n - 1$

- ❑ 32-bit number
 - 0000 0000 0000 = 0
 - 0000 0000 0001 = 1
 -
 - 1111 1111 1110 = 4,294,967,294
 - 1111 1111 1111 = 4,294,967,295 (4Billion)

Use hexadecimal notation
(binary notation is too long)

0x00000000
0x00000001

.....
0xffffffff
0xffffffff

Exercise #1

- ❑ Convert to 32-bit, 2's complement binary number : 56479
 - 56479/2 = 28239...1
 - 28239/2 = 14119...1
 - 14119/2 = 7059...1
 - 7059/2 = 3529...1
 - 3529/2 = 1764...1
 - 1764/2 = 882...0
 - 882/2 = 441...0
 - 441/2 = 220...1
 - 220/2 = 110...0
 - 110/2 = 55...0
 - 55/2 = 27...1
 - 27/2 = 13...1
 - 13/2 = 6...1
 - 6/2 = 3...0
 - 3/2 = 1...1

0000 0000 0000 0000
1101 1100 1001 1111
= 0000DC9F

Numbers

- ❑ Of course it gets more complicated:
 - numbers are finite (overflow)
 - fractions and real numbers
 - negative numbers
 - e.g., no subi instruction in MIPS; addi can add a negative number)

- ❑ How do we represent negative numbers?
 - i.e., which bit patterns will represent which numbers?

Possible Representations

❑ Sign Magnitude:	One's Complement	Two's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

$$-3 = \text{all } 1\text{'s} - 3 = 111 - 011 = 100$$

$$-3 = 2^n - 3 = (111+1) - 011 = 101$$

- ❑ Issues: balance, number of zeros, ease of operations
- ❑ Which one is best? Why?
 - 1's complement looks simpler because $(-a) = \text{invert } a$
 - 2's complement looks complex because $(-a) = \text{invert } a + 1$

Exercise #2

- ❑ Convert to 32-bit, 2's complement binary number : -2048

- 2048/2 = 1024...0
- 1024/2 = 512...0
- 512/2 = 256...0
- 256/2 = 128...0
- 128/2 = 64...0
- 64/2 = 32...0
- 32/2 = 16...0
- 16/2 = 8...0
- 8/2 = 4...0
- 4/2 = 2...0
- 2/2 = 1...0

0000 0000 0000 0000
0000 1000 0000 0000
=> (2's complement)

1111 1111 1111 1111
1111 0111 1111 1111 + 1
=
1111 1111 1111 1111
1111 1000 0000 0000
= FFFF800

Exercise #3

- ❑ Convert 2's complement binary number to hexadecimal and decimal:

- 1111 1111 1111 1111 1110 0011 1001

- ❑ To hexadecimal (don't care about sign)

- FFFFE39

- ❑ To decimal

- Sign : negative
- Magnitude : convert it first
 - 1111 1111 1111 1111 1110 0011 1001 => negate
 - 0000 0000 0000 0000 0000 0001 1100 0110 ±1 =>
 - 0000 0000 0000 0000 0000 0001 1100 0111
 - 1+2+4+64+128+256 =>
 - 455
- - 455

Possible Representations (Advanced Topic)

- ❑ With n-bit binary numbers (e.g. 3-bit, $2^n=8$)
 - Positive: $N < 2^{n-1}$ (e.g. 2=010)
 - Negative: -N (-2)
- ❑ Sign Magnitude
 - -N => sign bit=1, magnitude=10 => 110
- ❑ One's Complement
 - -N => $(2^n-1)-N = 2^n-N-1 = 8-2-1 = 5 = 101$
- ❑ Two's Complement
 - -N => $(2^n-1)-N + 1 = 2^n-N = 8-2 = 6 = 110$

Possible Representations

❑ Subtraction: a-b (a,b>0)

- One's Complement
 - If $a < b$: answer = $-(b-a)$, where $(b-a) > 0$
 - $a-b = a+(-b) = a+(2^n-b)-1 = 2^n-(b-a)-1$:
 - This is exactly the representation of $-(b-a)$: OK
 - If $a > b$: answer = $(a-b)$, where $(a-b) > 0$
 - $a-b = a+(-b) = a+(2^n-b)-1 = (a-b)+2^n-1 = (a-b)+8-1$:
 - This is not the representation of $(a-b)$: ???
 - 2^n is just ignored, but "-1" must be compensated

Ex)
$$\begin{array}{r} 0111 \ (7) \\ -0110 \ (6) \\ \hline 0001 \ (1) \end{array} \quad \Rightarrow \quad \begin{array}{r} 0111 \ (7) \\ +1001 \ (-6) \\ \hline 10000 \end{array} \rightarrow \text{ignore } 2^n \text{ (MSB) \& " +1" (one more add)}$$

*If 3-bit number.
8 cannot be represented in 3-bit, just goes away
But "-1" is a problem
=> must add "1" to get the answer (a-b)*

Possible Representations

❑ Subtraction: $a-b$ ($a,b>0$)

➤ Two's Complement

- If $a < b$: answer = $-(b-a)$, where $(b-a) > 0$
 - $a-b = a+(-b) = a+(2^n-b) = 2^n-(b-a)$:
 - This is exactly the representation of $-(b-a)$: OK
- If $a > b$: answer = $(a-b)$, where $(a-b) > 0$
 - $a-b = a+(-b) = a+(2^n-b) = (a-b)+2^n = (a-b)+8$:
 - This is not the representation of $(a-b)$, but 2^n is just ignored: OK
- Ex) 0111 (7) 0111 (7)
 -0110 (6) $\Rightarrow +1010$ (-6)

 0001 (1) $10001 \rightarrow$ ignore 2^n (MSB), that's it !!!

❑ That's why two's complement is mostly used in digital computers => Ease of subtraction !!!

MIPS

❑ 32 bit signed numbers:

```

0000 0000 0000 0000 0000 0000 0000 0000two = 0ten
0000 0000 0000 0000 0000 0000 0000 0001two = + 1ten
0000 0000 0000 0000 0000 0000 0000 0010two = + 2ten
...
0111 1111 1111 1111 1111 1111 1111 1110two = + 2,147,483,646ten
0111 1111 1111 1111 1111 1111 1111 1111two = + 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0000two = - 2,147,483,648ten
1000 0000 0000 0000 0000 0000 0000 0001two = - 2,147,483,647ten
1000 0000 0000 0000 0000 0000 0000 0010two = - 2,147,483,646ten
...
1111 1111 1111 1111 1111 1111 1111 1101two = - 3ten
1111 1111 1111 1111 1111 1111 1111 1110two = - 2ten
1111 1111 1111 1111 1111 1111 1111 1111two = - 1ten
    
```

maxint
minint

Two's Complement Operations

- ❑ Negating a two's complement number: invert all bits and add 1

- remember: “negate” and “invert” are quite different!

- ❑ Converting n bit numbers into numbers with more than n bits:

- MIPS 16 bit immediate gets converted to 32 bits for arithmetic

- copy the most significant bit (the sign bit) into the other bits

0010 -> 0000 0010

1010 -> 1111 1010

- "sign extension" (lbu vs. lb)

Signed vs. Unsigned Comparison

	Value?	
	2's comp	Unsigned?
R1= 0...00 0000 0000 0000 0001	1	1
R2= 0...00 0000 0000 0000 0010	2	2
R3= 1...11 1111 1111 1111 1111	-1	BIG

- ❑ After executing these instructions:

slt r4, r2, r1 ; if (r2 < r1) r4=1; else r4=0

slt r5, r3, r1 ; if (r3 < r1) r5=1; else r5=0

sltu r6, r2, r1 ; if (r2 < r1) r6=1; else r6=0

sltu r7, r3, r1 ; if (r3 < r1) r7=1; else r7=0

- ❑ What are values of registers r4 - r7? Why?

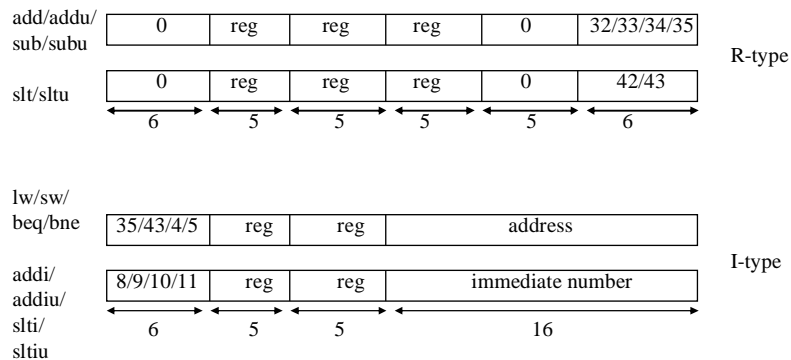
r4 = ; r5 = ; r6 = ; r7 = ;

MIPS Operations – i & u

- Arithmetic operations
 - Add – Addu, Addi, Addiu
 - Sub – Subu
 - slt – sltu, slti, sltiu
- Other arithmetic operations
 - Mult/Div
- Logical operations
 - And , Andi
 - Or , Ori
 - Xor , Xori
 - Nor
- Shift operations
 - SLT, SLTU, SLTI, SLTIU
 - SLL, SRL, SRA, SLLV, SRLV, SRAV

MIPS Instruction Types

Representing instructions in the computer



Unsigned numbers

- ❑ lb versus lbu (lh versus lhu)
 - Affects the entire word
 - lb: sign-extended, lbu: fill with zero's
- ❑ add versus addu
 - The only difference is that overflow is detected (exception) with add but not with addu
- ❑ addi versus addiu
 - addiu also sign-extends the immediate data before adding (YES, sign extends)
 - The only difference is that overflow is detected (exception) with addi but not with addiu

Unsigned numbers

0101010101010101 | 1111000011110000



	Cleared with 0's	Sign-extended	Overflow detected
lb		✓	
lbu	✓		
add			✓
addu			
addi		✓	✓
addiu		✓	
andi	✓		
ori	✓		
slti		✓	
sltiu		✓	

3.3 Addition & Subtraction

- ❑ Just like in grade school (carry/borrow 1s)

$$\begin{array}{r} 0101 \\ + 0001 \\ \hline \end{array} \quad \begin{array}{r} 0111 \\ - 0110 \\ \hline \end{array} \quad \begin{array}{r} 0110 \\ - 0101 \\ \hline \end{array}$$

- ❑ Two's complement operations easy

- subtraction using addition of negative numbers

$$\begin{array}{r} 0111 \\ + 1010 \\ \hline \end{array}$$

- ❑ Overflow (result too large for finite computer word):

- e.g., adding two n-bit numbers does not yield an n-bit number

$$\begin{array}{r} 0111 \\ + 0001 \\ \hline 1000 \end{array}$$

Detecting Overflow

- ❑ No overflow when adding a positive and a negative number
- ❑ No overflow when signs are the same for subtraction

- ❑ Overflow occurs when the value affects the sign:

- overflow when adding two positives yields a negative
- or, adding two negatives gives a positive
- or, subtract a negative from a positive and get a negative
- or, subtract a positive from a negative and get a positive

- ❑ An exception (interrupt) occurs

- Control jumps to predefined address for exception
- Interrupted address is saved for possible resumption

Appendix. Exception Type (cause)

0	Int	External interrupt (hardware)
4	AdEL	Address error (load or instruction fetch)
5	AdES	Address error (store)
6	IBE	Bus error on instruction fetch
7	DBE	Bus error on data load/store
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor <u>unimplemented</u>
12	Ov	<u>Arithmetic overflow</u> exception
13	Tr	Trap
15	FPE	Floating point exception

Appendix. Software Handler

□ Procedure

- Stop executing and
- Jump to fixed address 0x8000 0180

SPIM simulator uses 0x8000 0080

E.g.)

*CPU executes instruction whenever it is ON.
If the CPU is powered on, somebody has to
tell which address to start. (=reset address)
⇒0x0000 0000 is the natural choice (MIPS)
⇒0x000f fff0 for Intel CPU (ROM BIOS)*