Lecture 8: Number Crunching

- Today's topics:
  - MARS wrap-up
  - RISC vs. CISC
  - Numerical representations
  - Signed/Unsigned

Syllabus Reminders

School of Computing Guidelines

Class rosters are provided to the instructor with the student's legal name as well as "Preferred first name" (if previously entered by you in the Student Profile section of your CIS account). While CIS refers to this as merely a preference, I will honor you by referring to you with the name and pronoun that feels best for you in class, on papers, exams, etc. Please advise me of any name or pronoun changes (and please update CIS) so I can help create a learning environment in which you, your name, and your pronoun will be respected.

Syllabus Reminders

Cheating policy:

Working with others on assignments is a good way to learn the material and is encouraged. However, there are limits to the degree of cooperation that is permitted. Students may discuss among themselves the meaning of homework problems and possible approaches to solving them. Any written portion of an assignment, however, is to be done strictly on an individual basis. Note the School of Computing's Academic Misconduct Policy. BOTTOM LINE: You may not copy from another student or from any other source, and you may not allow another student to copy your work!! Any violation of the above is considered to be cheating and will result in a reduced or a failing grade in the class. TAs will be on the lookout for solution sets that appear very similar. Also, if your class rank in the assignments is significantly different from your class rank in the exams, only your rank in the exams will count towards your final grade.

Example Print Routine

directives & labels ⇒ assembler is sheltering you from low level details

syscalls

```
.data
  str:     .asciiz   "the answer is "
.text
  li     $v0, 4            # load immediate; 4 is the code for print_string
  la     $a0, str          #  the print_string syscall expects the string
                           #  address as the argument; la is the instruction
                           #  to load the address of the operand (str)
  syscall                  #  MARS will now invoke syscall-4
  li     $v0, 1            #  syscall-1 corresponds to print_int
  li     $a0, 5            #  print_int expects the integer as its argument
  syscall                  #  MARS will now invoke syscall-1
```

Example

- Write an assembly program to prompt the user for two numbers and print the sum of the two numbers

Example

```
.data
  str1:  .asciiz  "Enter 2 numbers:"
  str2:  .asciiz  "The sum is "

.text
  li   $v0, 4
  la   $a0, str1
  syscall
  li   $v0, 5
  syscall
  add  $t0, $v0, $zero
  li   $v0, 5
  syscall
  add  $t1, $v0, $zero
  li   $v0, 4
  la   $a0, str2
  syscall
  li   $v0, 1
  add  $a0, $t1, $t0
  syscall
```

read_int

print_int

IA-32 Instruction Set

RISC vs CISC

x86

conv

RISC μops

microarch

- Intel's IA-32 instruction set has evolved over 20 years – old features are preserved for software compatibility
- Numerous complex instructions – complicates hardware design (Complex Instruction Set Computer – CISC)
- Instructions have different sizes, operands can be in registers or memory, only 8 general-purpose registers, one of the operands is over-written
- RISC instructions are more amenable to high performance (clock speed and parallelism) – modern Intel processors convert IA-32 instructions into simpler micro-operations

Endian-ness

4B int

mem

4B

Two major formats for transferring values between registers and memory

lw

Memory: low address 45 7b 87 7f high address

Little-endian register: the first byte read goes in the low end of the register

Register: 7f 87 7b 45

Most-significant bit ↗ ↖ Least-significant bit (x86)

Big-endian register: the first byte read goes in the big end of the register

Register: 45 7b 87 7f

Most-significant bit ↗ ↖ Least-significant bit (MIPS, IBM)

Binary Representation

2^{31}

- The binary number

31st bit

32 b register

0th bit

2^{0}

01011000 00010101 00101110 11100111

Most significant bit

Least significant bit

$2^{32}-1$

4B

1×2^1

represents the quantity

$0 \times 2^{31} + 1 \times 2^{30} + 0 \times 2^{29} + \ldots + 1 \times 2^{0}$

carry

+1

-1

32nd bit

- A 32-bit word can represent 2^{32} numbers between 0 and 2^{32}-1

... this is known as the unsigned representation as we're assuming that numbers are always positive

ASCII Vs. Binary

- Does it make more sense to represent a decimal number in ASCII?
- Hardware to implement arithmetic would be difficult
- What are the storage needs? How many bits does it take to represent the decimal number 1,000,000,000 in ASCII and in binary?

ASCII Vs. Binary

- Does it make more sense to represent a decimal number in ASCII?
- Hardware to implement arithmetic would be difficult
- What are the storage needs? How many bits does it take to represent the decimal number 1,000,000,000 in ASCII and in binary?

  In binary: 30 bits (2^{30} > 1 billion)

  In ASCII: 10 characters, 8 bits per char = 80 bits

Negative Numbers

Signed

32 bits can only represent 2^{32} numbers – if we wish to also represent negative numbers, we can represent 2^{31} positive numbers (incl zero) and 2^{31} negative numbers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
…
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}\text{-}1$

positive → 2B

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2^{31}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -(2^{31} – 1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -(2^{31} – 2)$
…
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1$

-0
-1
-2
⋮
-2^{31}

2’s Complement

000 ... 010
111 111 ... 111
0000000000001

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2^{31}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -(2^{31} – 1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -(2^{31} – 2)$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1$

unsigned
$0 - 2^{32}-1$
$0 \sim 4B$

Signed
$-2^{31} \leftarrow 0 \rightarrow +2^{31}-1$
$-2B \sim 0 \sim +2B$

Why is this representation favorable?
Consider the sum of 1 and -2 .... we get -1
Consider the sum of 2 and -1 .... we get +1
This format can directly undergo addition without any conversions!

Each number represents the quantity
$x_{31}\ -2^{31} + x_{30}\ 2^{30} + x_{29}\ 2^{29} + \ldots + x_1\ 2^1 + x_0\ 2^0$

2's Complement

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
…
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2^{31}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -(2^{31} – 1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -(2^{31} – 2)$
…
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1$

Note that the sum of a number x and its inverted representation x' always equals a string of 1s (-1).

$x + x' = -1$

$x' + 1 = -x$ … hence, can compute the negative of a number by

$-x = x' + 1$ inverting all bits and adding 1

Similarly, the sum of x and –x gives us all zeroes, with a carry of 1

In reality, $x + (-x) = 2^n$ … hence the name 2's complement

Example

$-x = x' + 1$

- Compute the 32-bit 2's complement representations for the following decimal numbers:

  5, -5, -6

$\boxed{0 \cdots 00101}$ 5

-5

$\Downarrow$

$-5 = 1 \cdots 11011$ $5' + 1$

-6 $1 \cdots 11010$ $5'$

Example

- Compute the 32-bit 2's complement representations for the following decimal numbers:
  5, -5, -6

5: 0000 0000 0000 0000 0000 0000 0000 0101
-5: 1111 1111 1111 1111 1111 1111 1111 1011
-6: 1111 1111 1111 1111 1111 1111 1111 1010

Given -5, verify that inverting and adding 1 yields the number 5

Signed / Unsigned

- The hardware recognizes two formats:

  0 — 4B

  unsigned (corresponding to the C declaration unsigned int)
  -- all numbers are positive, a 1 in the most significant bit
  just means it is a really large number

  -2B ← 0 — +2B

  signed (C declaration is signed int or just int)
  -- numbers can be +/- , a 1 in the MSB means the number
  is negative

This distinction enables us to represent twice as many numbers when we're sure that we don't need negatives

MIPS Instructions

set on less than

Consider a comparison instruction:

slt $t0, $t1, $zero

and $t1 contains the 32-bit number 1111 01…01

What gets stored in $t0?

sltu $t0 = 0 (not less than)
unsigned

slt $t0 = 1 signed
(less than)

MIPS Instructions

Consider a comparison instruction:

slt $t0, $t1, $zero

and $t1 contains the 32-bit number 1111 01…01

What gets stored in $t0?

The result depends on whether $t1 is a signed or unsigned number – the compiler/programmer must track this and accordingly use either slt or sltu

```
slt   $t0, $t1, $zero     stores 1 in $t0
sltu  $t0, $t1, $zero     stores 0 in $t0
```

Sign Extension

addi __ $t1 __ 1100...

~~addit~~

$t1 32b

- Occasionally, 16-bit signed numbers must be converted into 32-bit signed numbers – for example, when doing an add with an immediate operand

111 16b const

- The conversion is simple: take the most significant bit and use it to fill up the additional bits on the left – known as sign extension

16b → 32b

So 2_{10} goes from 0000 0000 0000 0010 to
0000 0000 0000 0000 0000 0000 0000 0010

and -2_{10} goes from 1111 1111 1111 1110 to
1111 1111 1111 1111 1111 1111 1111 1110

look at 16th bit

Alternative Representations

- The following two (intuitive) representations were discarded because they required additional conversion steps before arithmetic could be performed on the numbers

  - sign-and-magnitude: the most significant bit represents +/- and the remaining bits express the magnitude

  - one's complement: -x is represented by inverting all the bits of x

Both representations above suffer from two zeroes