CS41B MACHINE

David Kauchak
CS 52 – Fall 2015

Admin

- Assignment 3
  - due Monday at 11:59pm
  - one small error in 5b (fast division) that's been fixed
- Midterm next Thursday in-class (10/1)
  - Comprehensive ☺
  - Closed books, notes, computers, etc.
  - ***Except***, may bring up to 2 pages of notes
  - Practice problems posted
    - Also some practice problems in the Intro SML reading
  - Midterm review sessions Tuesday and Wednesday (Q&A session for midterm)

Midterm topics

SML

recursion

math

Midterm topics

- Basic syntax
- SML built-in types
- Defining function
  - pattern matching
- Function type signatures
- Recursion!
- map
- exceptions
- Defining datatypes
- addition, subtraction, multiplication manually and on list digits
- Numbers in different bases
- Binary number representation (first part of today's lecture)
- *NOT CS41B material*

Binary number revisited

What number does 1001 represent in binary?

Depends!
Is it a signed number or unsigned?
If signed, what convention are we using?

Twos complement

For a number with *n* digits high order bit represents -2^{n-1}

unsigned	2^3	2^2	2^1	2^0
signed (twos complement)	$\boxed{-2^3}$	2^2	2^1	2^0

Twos complement

What number is it?

unsigned	1	0	0	1	9
	2^3	2^2	2^1	2^0	
signed (twos complement)	1	0	0	1	-7
	$\boxed{-2^3}$	2^2	2^1	2^0	

Twos complement

What number is it?

unsigned	1	1	1	1	15
	2^3	2^2	2^1	2^0	
signed (twos complement)	1	1	1	1	-1
	$\boxed{-2^3}$	2^2	2^1	2^0	

Twos complement

What number is it?

unsigned	1	1	0	0	12
	2^3	2^2	2^1	2^0	
signed (twos complement)	1	1	0	0	-4
	$\boxed{-2^3}$	2^2	2^1	2^0	

Twos complement

How many numbers can we represent with each approach using 4 bits?

16 (2^4) numbers, 0000, 0001, …., 1111
Doesn't matter the representation!

unsigned				
	2^3	2^2	2^1	2^0
signed (twos complement)				
	$\boxed{-2^3}$	2^2	2^1	2^0

Twos complement

How many numbers can we represent with each approach using 32 bits?

$2^{32} \approx$ 4 billion numbers

unsigned				
	2^3	2^2	2^1	2^0
signed (twos complement)				
	$\boxed{-2^3}$	2^2	2^1	2^0

Twos complement

What is the range of numbers we can represent for each approach?

unsigned: 0, 1, … 15
signed: -8, -7, …, 7

unsigned				
	2^3	2^2	2^1	2^0
signed (twos complement)				
	$\boxed{-2^3}$	2^2	2^1	2^0

binary representation	unsigned	
0000	0	
0001	1	
0010	?	
0011		
0100		
0101		
0110		
0111		
1000		
1001		
1010		
1011		
1100		
1101		
1110		
1111		

binary representation	unsigned	twos complement
0000	0	?
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	?
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	?
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

How can you tell if a number is negative?

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

High order bit!

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:

- Flip all of the bits (0 → 1 and 1→ 0)
- Add 1
- Resulting number is the magnitude of the original negative number

1101 $\xrightarrow{\text{flip the bits}}$ 0010 $\xrightarrow{\text{add 1}}$ 0011 → -3

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:

- Flip all of the bits (0 → 1 and 1→ 0)
- Add 1
- Resulting number is the magnitude of the original negative number

1110 $\xrightarrow{\text{flip the bits}}$ 0001 $\xrightarrow{\text{add 1}}$ 0010 → -2

Addition with twos complement numbers

```
  0001
+ 0101
------
    ?
```

Addition with twos complement numbers

```
     1
  0001
+ 0101
------
  0110
```

Addition with twos complement numbers

```
  0110
+ 0101
------
    ?
```

Addition with twos complement numbers

```
 1
 0110
+0101
 1011?
```

Addition with twos complement numbers

```
 1
 0110    6
+0101    5
 1011?   11
```

Overflow! We cannot represent this number (it's too large)

Addition with twos complement numbers

```
 0110
+1101
 ?
```

Addition with twos complement numbers

```
1 1
 0110
+1101
 0011
```

Addition with twos complement numbers

ignore the last carry

$$
\begin{array}{rl}
{}^{1\ \ 1} & \\
0110 & 6 \\
+\ 1101 & -3 \\
\hline
0011 & 3
\end{array}
$$

Subtraction

Ideas?

Subtraction

- Negate the 2nd number (flip the bits and add 1)
- Add them!

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:

- Flip all of the bits (0 → 1 and 1→ 0)
- Add 1
- Resulting number is the magnitude of the original negative number

Why does this work?

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:
- Flip all of the bits (0 → 1 and 1→ 0)
- Add 1
- Resulting number is the magnitude of the original negative number

What is:

1101 + 0010 ?

Hexadecimal numbers

Hexadecimal = base 16

What will be the digits?

Hexadecimal numbers

Hexadecimal = base 16

Digits

0
1
2
…
9
a (10)
b (11)
c (12)
d (13)
e (14)
f (15)

What number is 1ad?

Hexadecimal numbers

Hexadecimal = base 16

Digits

0
1
2
…
9
a (10)
b (11)
c (12)
d (13)
e (14)
f (15)

1	a	d	= 256+10*16+13=
16^2	16^1	16^0	= 429

Hexadecimal numbers

Hexadecimal = base 16

Digits

0
1
2
…
9
a (10)
b (11)
c (12)
d (13)
e (14)
f (15)

Hexadecimal is common in CS.
Why?

Hexadecimal numbers

Hexadecimal = base 16

Digits

0
1
2
…
9
a (10)
b (11)
c (12)
d (13)
e (14)
f (15)

Hexadecimal is common in CS.
4 bits = 1 hexadecimal digit

What is the following number in hex?

0110 1101 0101 0001

Hexadecimal numbers

Hexadecimal = base 16

Digits

0
1
2
…
9
a (10)
b (11)
c (12)
d (13)
e (14)
f (15)

Hexadecimal is common in CS.
4 bits = 1 hexadecimal digit

What is the following number in hex?

0110 1101 0101 0001

6 d 5 1

Computer internals

Computer internals simplified

CPU

What does it stand for?

What does it do?

RAM

What does it stand for?

What does it do?

Computer internals simplified

CPU

(**C**entral **P**rocessing **U**nit,
aka "*the processor*")

RAM

(**R**andom **A**ccess **M**emory,
aka "*memory*"
or "*main memory*")

Does all the work!

Temporary storage

Computer internals simplified

"the computer"

CPU

RAM

hard drive

Why do we need a hard drive?

Computer internals simplified

"the computer"

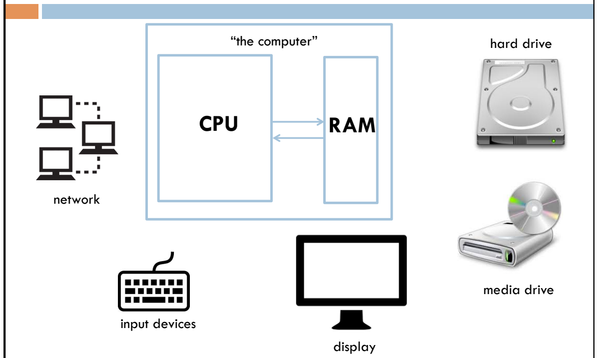
CPU

RAM

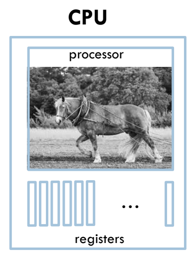
hard drive

- Persistent memory
- RAM only stores data while it has power

Computer simplified



Inside the CPU



processor: does the work

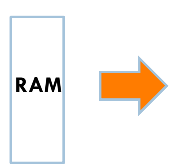
registers: local, fast memory slots

Why all these levels of memory?

Memory speed

operation	access time	times slower than register access	for comparison …
register	0.3 ns	1	1 s
RAM	120 ns	400	6 min
Hard disk	1ms	~million	1 month
google.com	0.4s	~billion	30 years

Memory

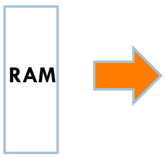


0101011110001010000010010 …

What is a byte?

?

Memory



$\underbrace{01010111}$ 10001010 00010010 …

byte = 8 bits
byte is abbreviated as B

My laptop has 16GB (gigabytes) of memory. How many bits is that?

Memory sizes

	bits
byte	8
kilobyte (KB)	2^10 bytes = ~8,000
megabyte (MB)	2^20 =~ 8 million
gigabyte (GB)	2^30 = ~8 billion

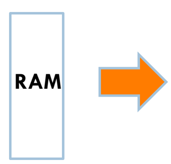
My laptop has 16GB (gigabytes) of memory. How many bits is that?

Memory sizes

	bits
byte	8
kilobyte (KB)	2^10 bytes = ~8,000
megabyte (MB)	2^20 =~ 8 million
gigabyte (GB)	2^30 = ~8 billion

~128 billion bits!

Memory



address	
0	01010111
1	10001010
2	00010010
3	01011010
…	…

Memory is byte addressable

Memory

RAM

address	
0	01010111
1	10001010
2	00010010
3	01011010
…	…

Memory is organized into “words”, which is the most common functional unit

Memory

RAM

address	32-bit words
0	10101011 10001010 00010010 01011010
4	11001011 00001110 01010010 01010110
8	10111011 10010010 00000000 01110100
…	…

Most modern computers use 32-bit (4 byte) or 64-bit (8 byte) words

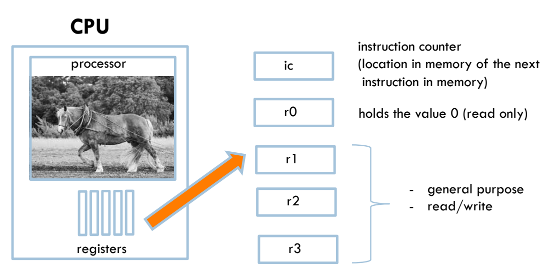
Memory in the CS41B Machine

RAM

address	16-bit words
0	10101011 10001010
2	00010010 01011010
4	11001011 00001110
…	…

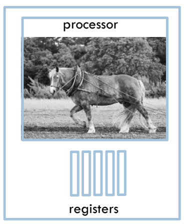
We’ll use 16-bit words for our model (the CS41B machine)

CS41B machine



CS41B instructions

CPU



What types of operations might we want to do (think really basic)?

abbreviation	arguments	action
	Register Instructions	
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

abbreviation	arguments	action
	Register Instructions	
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

operation name
(always three characters)

abbreviation	arguments	action
	Register Instructions	
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

operation arguments
R = register (e.g. r0)
S = signed number (byte)

abbreviation	arguments	action
Register Instructions		
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

operation function
dest = first register
src0 = second register
src1 = third register
arg = number/argument

add r1 r2 r3

What does this do?

abbreviation	arguments	action
Register Instructions		
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

add r1 r2 r3

r1 = r2 + r3

Add contents of registers r2 and r3 and store the result in r1

abbreviation	arguments	action
Register Instructions		
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

adc r2 r1 10

What does this do?

abbreviation	arguments	action
Register Instructions		
mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

adc r2 r1 10

r2 = r1 + 10

Add 10 to the contents of register r1 and store in r2

abbreviation	arguments	action

Register Instructions

mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

adc r1 r0 8
neg r2 r1
sub r2 r1 r2

What number is in r2?

abbreviation	arguments	action

Register Instructions

mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg

adc r1 r0 8 — r1 = 8
neg r2 r1 — r2 = -8, r1 = 8
sub r2 r1 r2 — r2 = 16

abbreviation	arguments	action

Register Instructions

mov	RR-	dest = src0
neg	RR-	dest = −src0
add	RRR	dest = src0 + src1
sub	RRR	dest = src0 − src1
adc	RRS	dest = src0 + arg
sbc	RRS	dest = src0 − arg