CSE 410
Computer Systems

Hal Perkins

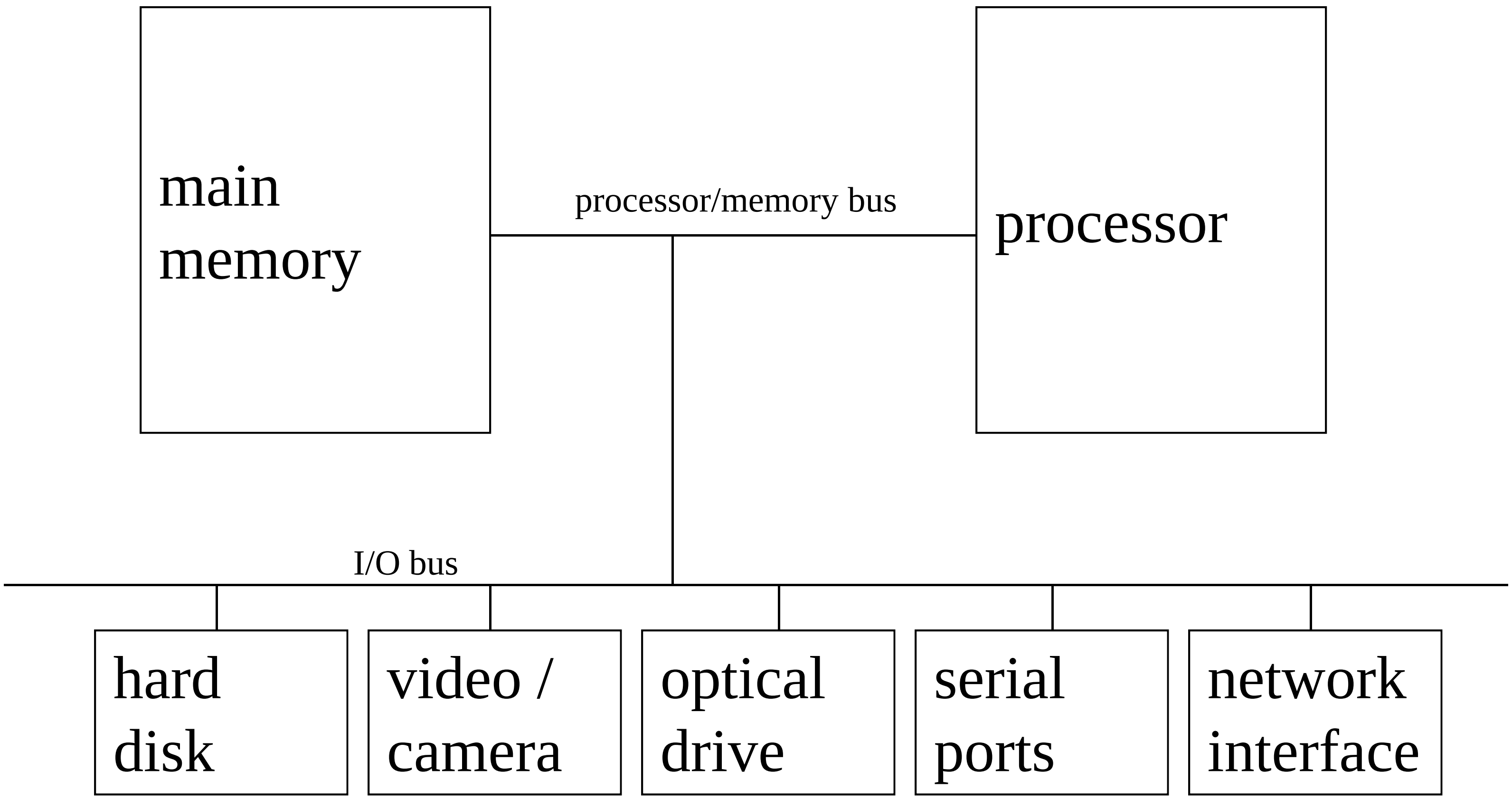
Spring 2010

Lecture 2 – Information Representation

Reading and References

- Reading
  - Computer Organization and Design, Patterson and Hennessy
    - Chapter 2, sec. 2.4, 2.9 (first page only for now)

Review: A Computer is …



Processor & Memory

- For now we focus on the processor & memory
- Processor operation
  – Fetch next instruction from memory
  – Fetch instruction operands (data) from memory
  – Perform operation (add, subtract, test, …)
  – Store result (if any) in memory
  – Repeat
    - Billions of times a second
- Memory holds all instructions and data
- What is memory made of?

Bits

- All memories are composed of (billions of) bits
- A bit is:
  - high or low voltage
  - 0 or 1
  - true or false
  - yes or no
  - on or off

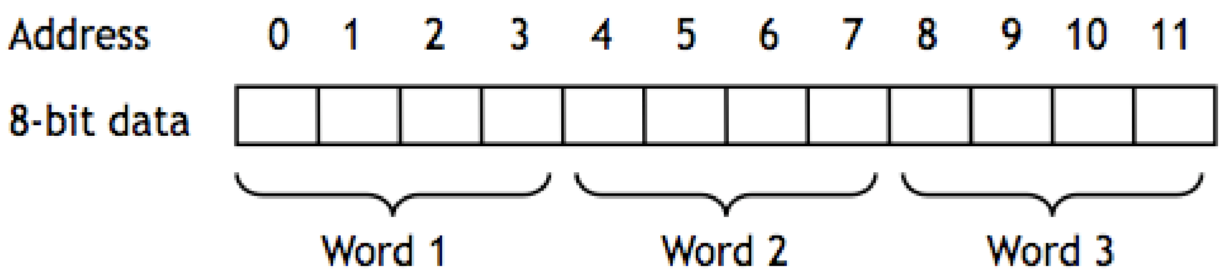
  It’s all how you interpret it
- But to store anything complicated we use a bunch of bits to make up a number, character, instruction, …

Computer Memory

- All memories are organized by grouping sets of bits into individual memory cells
- Each cell has an *address* and its *contents*
- Standard organization now: 1 cell = 8 bits = 1 *byte*
- A single byte can hold
  - A small integer (0-255 or -128-127)
  - A single character (‘a’, ‘A’, ‘?’, ‘#’, ‘ ’, …)
  - A boolean value (00000000, 00000001)

Memory Organization

- Memory is a linear array of bytes; each byte has an address (or location) – *not* the same as its contents



- Groups of bytes can be treated as a single unit

Some common storage units

unit	*# bits*
byte	8
half-word	16
word	32
double word	64

- Terminology varies: this is how MIPS does it; the Intel x86 calls 16 bits a word & 32 bits a double-word

Alignment

- An object in memory is “aligned” when its address is a multiple of its size
- Byte: always aligned
- Halfword: address is multiple of 2
- Word: address is multiple of 4
- Double word: address is multiple of 8
- Alignment simplifies load/store hardware
  – And is required by MIPS, but not x86

Binary Arithmetic

- Just as we do with decimal numbers, we can treat a collection of bits as a multi-digit number in base 2

- Example: $1010_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

  = ______________

- You try it: 11001 = ______________ ?

Binary, Hex, and Decimal

- It's unwieldy to work with long strings of binary digits, so we group them in chunks of 4 and treat each chunk as a digit in base 16
- Hex digits:
  - 0 = 0000, 1 = 0001, 2 = 0010, 3 = 0011
  - 4 = 0100, 5 = 0101, 6 = 0110, 7 = 0111
  - 8 = 1000, 9 = 1001, __ = 1010, __ = 1011
  - __ = 1100, __ = 1101, __ = 1110, __ = 1111

- Usual notation for hex integer in C, Java, …: 0x1c4

Hex Numbers

- What is 0x2a5 in decimal?
  - 0x2a5 = $2 \times 16^2 + a \times 16^1 + 5 \times 16^0$

    = ______________________________

- What about 0xbad?

  ____

- Be sure you realize that 0x11 $\neq 11_{10} \neq 11_2$

More problems

- What is 605_{10} in hex?

- What is 0xbeef in binary?
  - (Hint: there's a trick)

Binary, Hex, and Decimal

Binary_2	Hex_{16}	$10^3=1000_{10}$	$10^2=100_{10}$	$10^1=10_{10}$	$10^0=1_{10}$
11	0x3				3
1001	0x9				9
1010	0xA			1	0
1111	0xF			1	5
1 0000	0x10			1	6
1 1111	0x1F			3	1
111 1111	0x7F		1	2	7
1111 1111	0xFF		2	5	5

Binary, Hex, and Decimal

$2^8=256_{10}$	$2^7=128_{10}$	$2^6=64_{10}$	$2^5=32_{10}$	$2^4=16_{10}$	$2^3=8_{10}$	$2^2=4_{10}$	$2^1=2_{10}$	$2^0=1_{10}$	Hex_{16}	$Decimal_{10}$
							1	1	0x3	3
					1	0	0	1	0x9	9
					1	0	1	0	0xA	10
					1	1	1	1	0xF	15
				1	0	0	0	0	0x10	16
				1	1	1	1	1	0x1F	31
		1	1	1	1	1	1	1	0x7F	127
	1	1	1	1	1	1	1	1	0xFF	255

Binary, Hex, and Decimal

$Binary_2$	$16^4=65536_{10}$	$16^3=4096_{10}$	$16^2=256_{10}$	$16^1=16_{10}$	$16^0=1_{10}$	$Decimal_{10}$
11					3	3
1001					9	9
1010					A	10
1111					F	15
1 0000				1	0	16
1 1111				1	F	31
111 1111				7	F	127
1111 1111				F	F	255

Unsigned binary numbers

- Each bit represents a power of 2
- For unsigned numbers in a fixed width n-bit field:
  - 2^n distinct values
  - the minimum value is 0
  - the maximum value is 2^{n-1}, where n is the number of bits in the field
- Fixed field widths determine many limits
  - 5 bits = 32 possible values ($2^5 = 32$)
  - 10 bits = 1024 possible values ($2^{10} = 1024$)

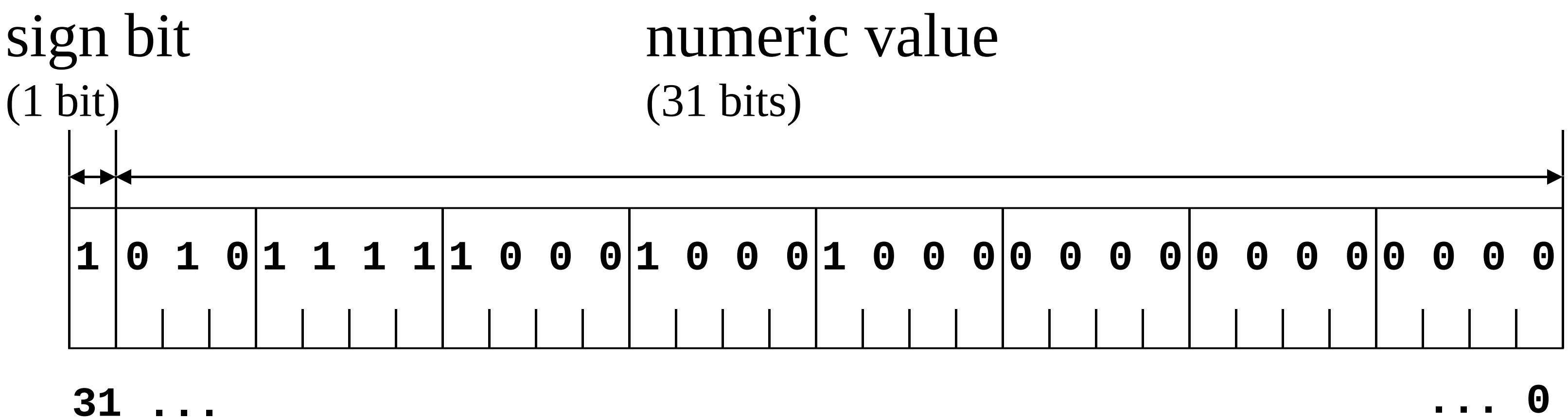
Signed Numbers

- For unsigned numbers,
  – each bit position represents a power of 2
  – range of values is 0 to 2^n-1
- How can we indicate negative values?
  – two states: positive or negative
  – a binary bit indicates one of two states: 0 or 1
  $\Rightarrow$ use one bit for the sign bit

Where is the sign bit?

- Could use an additional bit to indicate sign
  - each value would require 33 bits
  - would really foul up the hardware design
- Could use any bit in the 32-bit word
  - any bit but the left-most (high order) would complicate the hardware tremendously
- ∴The high order bit (left-most) is the sign bit
  - remaining bits indicate the value

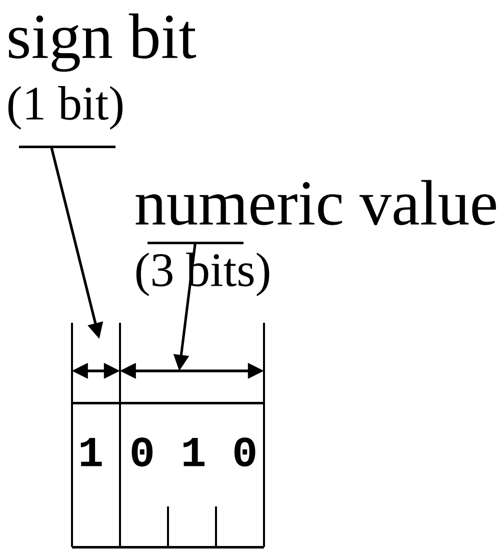
Format of 32-bit signed integer



- Bit 31 is the sign bit
  - 0 for positive numbers, 1 for negative numbers
  - aka most significant bit (msb), high order bit

Example: 4-bit signed numbers

Hex	Bin	Unsigned Decimal	Signed Decimal
F	1111	15	-1
E	1110	14	-2
D	1101	13	-3
C	1100	12	-4
B	1011	11	-5
A	1010	10	-6
9	1001	9	-7
8	1000	8	-8
7	0111	7	7
6	0110	6	6
5	0101	5	5
4	0100	4	4
3	0011	3	3
2	0010	2	2
1	0001	1	1
0	0000	0	0



Two's complement notation

- Note special arrangement of negative values
- One zero value, one extra negative value
- The representation is exactly what you get by doing a subtraction

Decimal	Binary
1	0001
- 7	- 0111
----	----
-6	1010

Why "two's" complement?

- In an n-bit binary word, negative x is represented by the value of 2^n-x. The radix (or base) is 2.
- 4-bit example

  $2^4 = 16$. What is the representation of -6?

```
Decimal        Binary

   16          10000
-   6       -   0110
----            ----
   10           1010
```

Negating a number

- Given x, how do we represent negative x?

```
          negative(x) = 2^n-x
and x+complement(x) = 2^n-1
so        negative(x) = 2^n-x = complement(x)+1
```

- The easy shortcut
  - write down the value in binary
  - complement all the bits
  - add 1

Example: the negation shortcut

```
 decimal 6 = 0110 = +6
complement = 1001
     add 1 = 1010 = -6

decimal -6 = 1010 = -6
complement = 0101
     add 1 = 0110 = +6
```

Why 2's complement? (Again)

- The key advantage is that we can add two numbers together without paying any attention to the sign and we get the properly signed result

What About Non-Numeric Data?

- Everything is bits
- So assign (arbitrarily) specific bit patterns (numbers) to represent different characters
- Two most common codes
  - ASCII – original 7-bit code, early 1960's
    - How many possible characters?
  - Unicode – 16- to 32-bit code to represent enough different characters to encode all currently used alphabets; started in the late 1980s

ASCII

32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	I	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	del

But wait, there's more!

- We still haven't looked at
  - Floating-point numbers (scientific notation)
  - Strings/arrays
  - Records/structs/objects
  - Colors and images
  - Sounds
- We'll see some of this, but it's all encoded as numbers (i.e., collections of bits)
- But next we need to look at how the computer processes these things – instructions