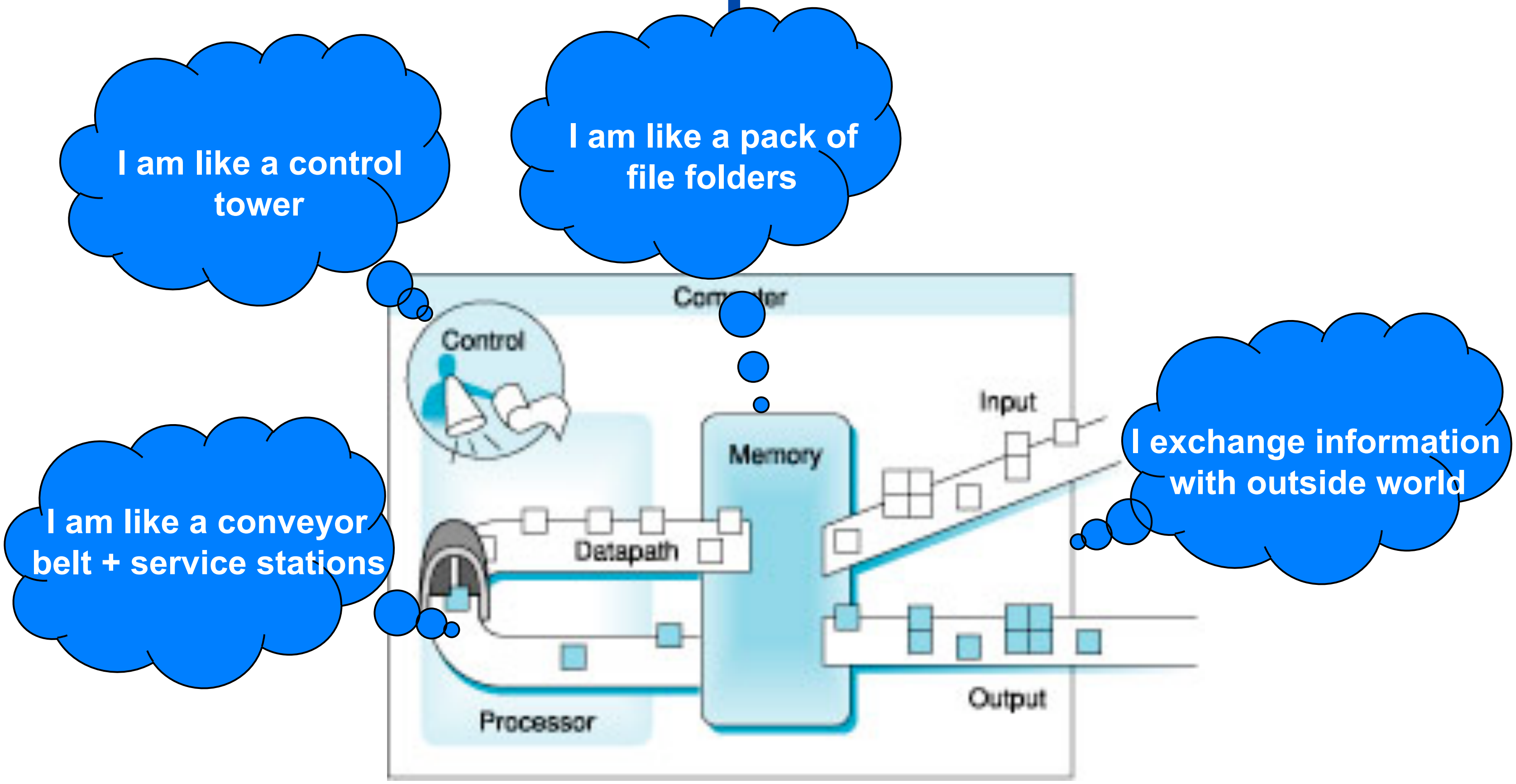
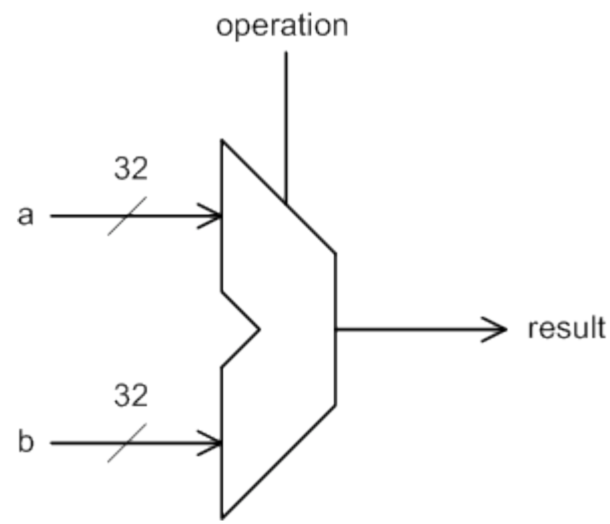
Five classic components



Binary arithmetic

- (Sounds scary)
- So far we studied
  - Instruction set architecture basic
  - MIPS architecture & assembly language
- We will review binary arithmetic algorithms and their implementations
- Binary arithmetic will form the basis for CPU's datapath design



Binary number representations

- We looked at how to represent a number (in fact the value represented by a number) in binary
  - Unsigned numbers – everything is positive

- We will deal with more complicated cases
  - Negative numbers
  - Real numbers (a.k.a. floating-point numbers)

Unsigned Binary Numbers

- Limited number of binary numbers (patterns of 0s and 1s)
  - 8-bit number: 256 patterns, 00000000 to 11111111
  - in general, there are 2^N bit patterns, where N is bit width

    16 bit: 2^{16} = 65,536 bit patterns

    32 bit: 2^{32} = 4,294,967,296 bit patterns

- Unsigned numbers use patterns for ***0*** and ***positive numbers***
  - 8-bit number range [0..255] corresponds to

00000000	0
00000001	1
...	...
11111111	255

  - 32-bit number range [0..4294,967,295]
  - in general, the range is [0..2^N-1]

Unsigned Binary Numbers

- Binary addition
  - `0 + 0 = 0, carry = 0 (no carry)`
  - `1 + 0 = 1, carry = 0`
  - `0 + 1 = 1, carry = 0`
  - `1 + 1 = 0, carry = 1`

- Binary subtraction
  - `0 - 0 = 0, borrow = 0 (no borrow)`
  - `1 - 0 = 1, borrow = 0`
  - `0 - 1 = 1, borrow = 1`
  - `1 - 1 = 0, borrow = 0`

Unsigned Binary Numbers

- Binary arithmetic is straightforward

- Addition: Just add numbers and carry as necessary
- Consider adding 8-bit numbers:

carry overflowed

carry

```
 01001111x                            11101111x
  01101011       107d                  11101011       235d
+ 01001101        77d                + 01001101        77d
----------       ----                ----------       ----
  10111000       184d                 100111000       312d
```

legal number: betw. 0 and 255

illegal number: overflowed 8 bits

Unsigned Binary Numbers

- Binary arithmetic is straightforward

- Subtraction: Just subtract and borrow as necessary
- Consider subtracting 8-bit numbers:

borrow

```
   111                                 111111
  01101011      107d                    01101011      107d
- 01001101       77d                  - 01101101      109d
----------      ----                  ----------      ----
  00011110       30d                   111111110      -2d
```

legal number: betw. 0 and 255

illegal number: underflowed 8 bits (i.e., “borrow overflow”)

Unsigned Binary to Decimal

- How to convert binary number?
  - First, each digit is position *i*, numbered right to left
  - e.g., for 8-bit number: $b_7 b_6 b_5 b_4\ b_3 b_2 b_1 b_0$
- Now, we just add up powers of 2
  - $b_0 \times 2^0 + b_1 \times 2^1 + b_2 \times 2^2 + \ldots + b_7 \times 2^7$
- An example

  1011 0111

  $= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 1 \times 2^7$

  $= 1 + 2 + 4 + 0 + 16 + 32 + 0 + 128$

  = 183d
- $v = \sum (b_i \times 2^i)$, where $0 \leq i \leq K-1$, where K=# bits, i is bit posn

Unsigned Binary Numbers in MIPS

- MIPS instruction set provides support
  - addu $1,$2,$3 - adds two unsigned numbers ($2,$3)
  - addiu $1,$2,10 - adds unsigned number with **signed** immediate
  - subu $1,$2,$3 - subtracts two unsigned numbers
  - etc.

- Primary issue: **The carry/borrow out is ignored**
  - Overflow is possible, but it is ignored
  - Signed versions take special action on overflow (we'll see shortly!)

- Unsigned memory accesses: lbu, lhu
  - Loaded value is treated as unsigned number
  - Convert from smaller bit width (8 or 16) to a 32-bit number
  - **Upper bits in the 32-bit destination register are set to 0s**

Important 7-bit Unsigned Numbers

- American Standard Code for Information Interchange (ASCII)
  - Developed in early 60s, rooted in telecomm
  - Maps 128 bit patterns (2^7) into control, alphabet, numbers, graphics
  - Provides control values present in other important codes (at the time)
  - 8^{th} bit might be present and used for error detection (parity)

- Control: Null (0), Bell (7), BS (8), LF (0A), CR (0D), DEL (7F)
- Numbers: (30-39)
- Alphabet: Uppercase (41-5A), Lowercase (61-7A)
- Other (punctuation, etc): 20-2F, 3A-40, 5E-60, 7B-7E

- Unicode: A larger (8,16,32 bit) encoding; backward compatible with ASCII

Regular ASCII Chart (character codes 0 - 127)

000d	00h		(nul)	016d	10h	►	(dle)	032d	20h	sp	048d	30h	0	064d	40h	@	080d	50h	P	096d	60h	`	112d	70h	p
001d	01h	☺	(soh)	017d	11h	◄	(dc1)	033d	21h	!	049d	31h	1	065d	41h	A	081d	51h	Q	097d	61h	a	113d	71h	q
002d	02h	☻	(stx)	018d	12h	↕	(dc2)	034d	22h	"	050d	32h	2	066d	42h	B	082d	52h	R	098d	62h	b	114d	72h	r
003d	03h	♥	(etx)	019d	13h	‼	(dc3)	035d	23h	#	051d	33h	3	067d	43h	C	083d	53h	S	099d	63h	c	115d	73h	s
004d	04h	♦	(eot)	020d	14h	¶	(dc4)	036d	24h	$	052d	34h	4	068d	44h	D	084d	54h	T	100d	64h	d	116d	74h	t
005d	05h	♣	(enq)	021d	15h	§	(nak)	037d	25h	%	053d	35h	5	069d	45h	E	085d	55h	U	101d	65h	e	117d	75h	u
006d	06h	♠	(ack)	022d	16h	▬	(syn)	038d	26h	&	054d	36h	6	070d	46h	F	086d	56h	V	102d	66h	f	118d	76h	v
007d	07h	•	(bel)	023d	17h	↨	(etb)	039d	27h	'	055d	37h	7	071d	47h	G	087d	57h	W	103d	67h	g	119d	77h	w
008d	08h	◘	(bs)	024d	18h	↑	(can)	040d	28h	(	056d	38h	8	072d	48h	H	088d	58h	X	104d	68h	h	120d	78h	x
009d	09h		(tab)	025d	19h	↓	(em)	041d	29h	)	057d	39h	9	073d	49h	I	089d	59h	Y	105d	69h	i	121d	79h	y
010d	0Ah		(lf)	026d	1Ah		(eof)	042d	2Ah	*	058d	3Ah	:	074d	4Ah	J	090d	5Ah	Z	106d	6Ah	j	122d	7Ah	z
011d	0Bh	♂	(vt)	027d	1Bh	←	(esc)	043d	2Bh	+	059d	3Bh	;	075d	4Bh	K	091d	5Bh	[	107d	6Bh	k	123d	7Bh	{
012d	0Ch	♀	(np)	028d	1Ch	∟	(fs)	044d	2Ch	,	060d	3Ch	<	076d	4Ch	L	092d	5Ch	\	108d	6Ch	l	124d	7Ch	|
013d	0Dh		(cr)	029d	1Dh	↔	(gs)	045d	2Dh	-	061d	3Dh	=	077d	4Dh	M	093d	5Dh	]	109d	6Dh	m	125d	7Dh	}
014d	0Eh	♪	(so)	030d	1Eh	▲	(rs)	046d	2Eh	.	062d	3Eh	>	078d	4Eh	N	094d	5Eh	^	110d	6Eh	n	126d	7Eh	~
015d	0Fh	☼	(si)	031d	1Fh	▼	(us)	047d	2Fh	/	063d	3Fh	?	079d	4Fh	O	095d	5Fh	_	111d	6Fh	o	127d	7Fh	⌂

Extended ASCII Chart (character codes 128 – 255; Codepage 850)

128d	80h	Ç	144d	90h	É	160d	A0h	á	176d	B0h	░	192d	C0h	└	208d	D0h	ð	224d	E0h	Ó	240d	F0h	-
129d	81h	ü	145d	91h	æ	161d	A1h	í	177d	B1h	▒	193d	C1h	┴	209d	D1h	Ð	225d	E1h	ß	241d	F1h	±
130d	82h	é	146d	92h	Æ	162d	A2h	ó	178d	B2h	▓	194d	C2h	┬	210d	D2h	Ê	226d	E2h	Ô	242d	F2h	‗
131d	83h	â	147d	93h	ô	163d	A3h	ú	179d	B3h	│	195d	C3h	├	211d	D3h	Ë	227d	E3h	Ò	243d	F3h	¾
132d	84h	ä	148d	94h	ö	164d	A4h	ñ	180d	B4h	┤	196d	C4h	─	212d	D4h	È	228d	E4h	õ	244d	F4h	¶
133d	85h	à	149d	95h	ò	165d	A5h	Ñ	181d	B5h	Á	197d	C5h	┼	213d	D5h	ı	229d	E5h	Õ	245d	F5h	§
134d	86h	å	150d	96h	û	166d	A6h	ª	182d	B6h	Â	198d	C6h	ã	214d	D6h	Í	230d	E6h	µ	246d	F6h	÷
135d	87h	ç	151d	97h	ù	167d	A7h	º	183d	B7h	À	199d	C7h	Ã	215d	D7h	Î	231d	E7h	þ	247d	F7h	
136d	88h	ê	152d	98h	ÿ	168d	A8h	¿	184d	B8h	©	200d	C8h	╚	216d	D8h	Ï	232d	E8h	Þ	248d	F8h	°
137d	89h	ë	153d	99h	Ö	169d	A9h	®	185d	B9h	╣	201d	C9h	╔	217d	D9h	┘	233d	E9h	Ú	249d	F9h	¨
138d	8Ah	è	154d	9Ah	Ü	170d	AAh	¬	186d	BAh	║	202d	CAh	╩	218d	DAh	┌	234d	EAh	Û	250d	FAh	·
139d	8Bh	ï	155d	9Bh	ø	171d	ABh	½	187d	BBh	╗	203d	CBh	╦	219d	DBh	█	235d	EBh	Ù	251d	FBh	¹
140d	8Ch	î	156d	9Ch	£	172d	ACh	¼	188d	BCh	╝	204d	CCh	╠	220d	DCh	▄	236d	ECh	ý	252d	FCh	²
141d	8Dh	ì	157d	9Dh	Ø	173d	ADh	¡	189d	BDh	¢	205d	CDh	═	221d	DDh	¦	237d	EDh	Ý	253d	FDh	³
142d	8Eh	Ä	158d	9Eh	×	174d	AEh	«	190d	BEh	¥	206d	CEh	╬	222d	DEh	Ì	238d	EEh	¯	254d	FEh	■
143d	8Fh	Å	159d	9Fh	ƒ	175d	AFh	»	191d	BFh	┐	207d	CFh	¤	223d	DFh	▀	239d	EFh	´	255d	FFh	

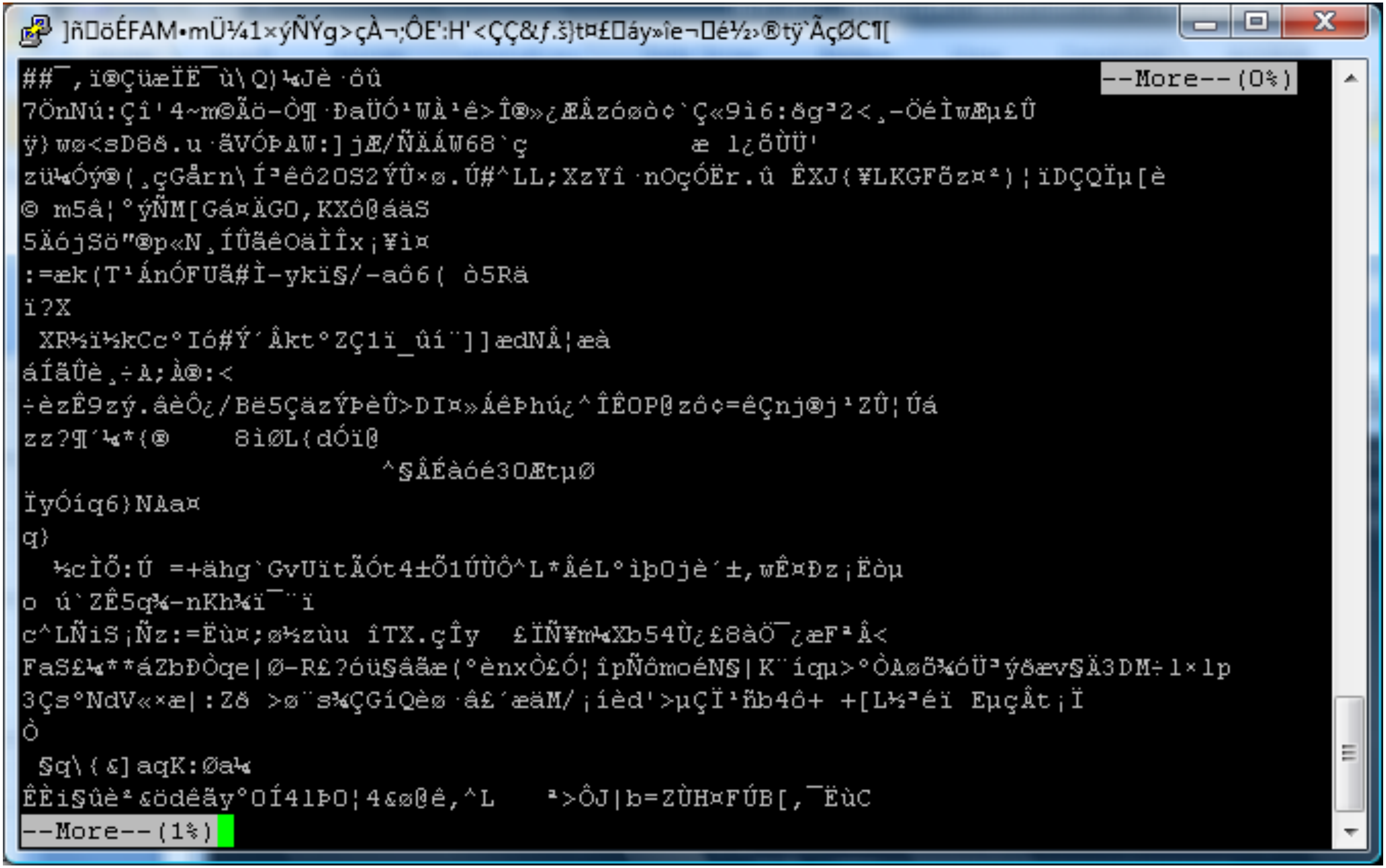
Hexadecimal to Binary

0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Groups of ASCII-Code in Binary

Bit 6	Bit 5	Group
0	0	Control Characters
0	1	Digits and Punctuation
1	0	Upper Case and Special
1	1	Lower Case and Special

Michael Goerz, Sept 04 2000



OOPS!!! (listed object file)

Signed Numbers

- How to represent positive and ***negative*** numbers?
- We still have a limited number of bit patterns
  - 8-bit: 256 bit patterns (i.e., 00000000 … 11111111)
  - 16 bit: 2^{16} = 65,536 bit patterns
  - 32 bit: 2^{32} = 4,294,967,296 bit patterns
- Re-assign bit patterns differently
  - Some patterns are assigned to negative numbers, some to positive
- How to assign available patterns? Three ways:
  - Sign magnitude, 1's complement, 2's complement

Method 1: sign-magnitude

- Same method we use for decimal numbers
- {sign bit, absolute value (magnitude)}
  - Sign bit (msb): 0 – positive, 1 – negative
  - Examples, assume 4-bit representation
    - 0000 +0
    - 0011 +3
    - 1001 -1
    - 1111 -7
    - 1000 -0 (two 0’s???)
- Properties
  - Two 0s – a positive 0 and a negative 0?
  - Equal # of positive and negative numbers
  - A + (-A) does not give zero!
  - Consider sign during arithmetic

Sign-magnitude

- Let’s check A + (-A) is not zero
- Consider N = 5 bits number. Zero is 00000 or 10000.
- Try this: -4 + 4 = ?????

```
-4 is 10100
 4 is 00100

so, let's add them together:
  10100  -4d
+ 00100   4d
-------  ---
  11000  -8d  YIKES!
```

Method 2: one's complement

- Negation of +X is $((2^N - 1) - X)$, where N is number of bits
  - $A + (-A) = 2^N - 1$ (i.e., -0)
  - Given a number A, it's negation is done by (1111…1111 – A)
  - In fact, simple bit-by-bit inversion will give the same-magnitude number with a different sign
  - Examples, assume 4-bit representation
    - 0000
    - 0011
    - 1001
    - 1111 -0
    - 1000
- Properties
  - There are two 0s
  - There are equal # of positive and negative numbers
  - A+(-A) = 0 (whew!) but… A+0=A only works for +0 (try it with -0!)
  - 2 step process for subtraction (accounts for "carry out")

One's Complement

- Negation of X $(2^N – 1) – X)$, positive are usual value
- Consider N=4

Binary	One's	Binary	One's
0000	0	1000	-7
0001	1	1001	-6
0010	2	1010	-5
0011	3	1011	-4
0100	4	1100	-3
0101	5	1101	-2
0110	6	1110	-1
0111	7	1111	-0

notice how the counting works: 1111 is -0… then -1… -2… etc.

One’s Complement

- Let’s check the “0 property”: A + (-A) = 0
- Suppose A = 5

5 is 0101

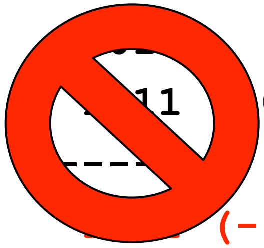
negation of 5 is $(2^4-1)-5 = (16-1) - 5 = 15 - 5 = 10$

10 (unsigned) is 1010

check the table: 1010 is -5 in 1’s complement

now, let’s try 5 + (-5) in 1’s complement

```
  0101          5        1010
+ 1010         -5      + 0000 (+0)        11    0)
------       ----      ------
  1111         -0        1010 (-5)             (-6)
```



Method 3: two's complement

- Negation is $(2^N – X)$
  - $A + (-A) = 2^N$
  - Given a number A, it's negation is done by (1111…1111 – A) + 1
  - In fact, simple bit-by-bit inversion followed by adding 1 will give the same-magnitude number with a different sign
  - Examples, assume 4-bit representation
    - 0000
    - 0011
    - 1001
    - 1111
    - 1000 ?
- Properties
  - There is a single 0
  - There are unequal # of positive and negative numbers
  - Subtraction is simplified - one step based on addition (we'll see! ☺)

Two's Complement

- Negation of X (2^N – X), positive are usual value
- Consider N=4

Binary	One's	Binary	One's
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1

notice how the counting works: 1000 is -8… 1001 is -7… etc.

Two's Complement

- Let's check the "0 property": A + (-A) = 0
- Suppose A = 5

```
5 is 0101
negation of 5 is 2^4 - 5 = 16 - 5 = 11
11(unsigned) is 1011
check the table: 1011 is -5 in 2's complement
now, let's try 5 + (-5) in 2's complement

  0101            5            1011             0111   (7)
+ 1011           -5          + 0000  (0)      + 0001   (1)
------         ----          ------           ------
1 0000            0            1011  (-5)       1000  (-8)
```

Two’s Complement

- Negation: (2^8 – X) vs. (11111111 - X) + 1
- Note 2^8 needs 9 bits:
  - 2^8 is 256, from earlier conversion process: 1 0000 0000 = 1 * 2^8
- Whereas the other form has only 8 bits. Let’s try it!
  - Consider X = 10 and we want to find -10

```
  1111 1111
- 0000 1010   (10d)
  ---------
  1111 0101 (-11d)
+         1
  ---------
  1111 0110 (-10d)
```

Oh, cool!
That’s just flipping bits!

Two's Complement

- How to convert binary 2's complement number?
  - Same as before, except most significant bit is "sign"
- Consider an 8-bit 2's complement number
  - $b_0 \times 2^0 + b_1 \times 2^1 + b_2 \times 2^2 + \ldots + \mathbf{b_7 \times (-2^7)}$
- An example

  1011 0111

  $= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + \mathbf{1 \times (-2^7)}$

  $= 1 + 2 + 4 + 0 + 16 + 32 + 0 + \mathbf{(-128)}$

  = -73d
- What is 73d in 2's complement binary number?

- $v = (\sum (b_i \times 2^i)) + b_{K-1} \times -2^{K-1}$,

  where $0 \leq i < K-1$, where K=# bits, i is bit posn