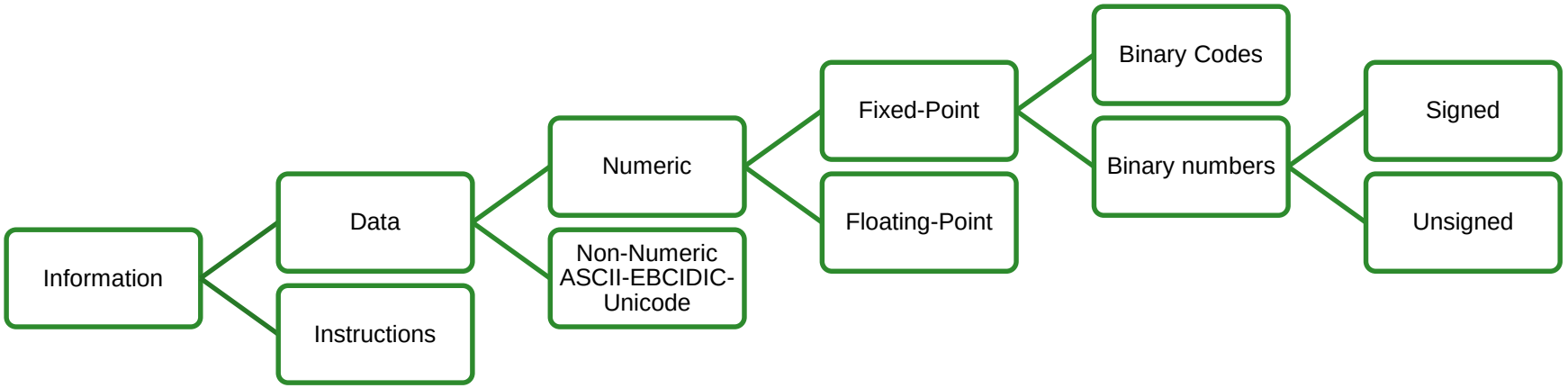
COMPUTER ARITHMETIC

Computer Arithmetic

Information - data



Numeric data

- Binary numbers
- Binary Codes
- Unsigned and Signed Magnitude numbers
- Fixed-Point numbers
- Floating-Point numbers

Binary numbers

$(54.5)_{10} = (110110.1)_2$

Why?

2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
1	**1**	**0**	**1**	**1**	**0**	●	**1**	**0**	...

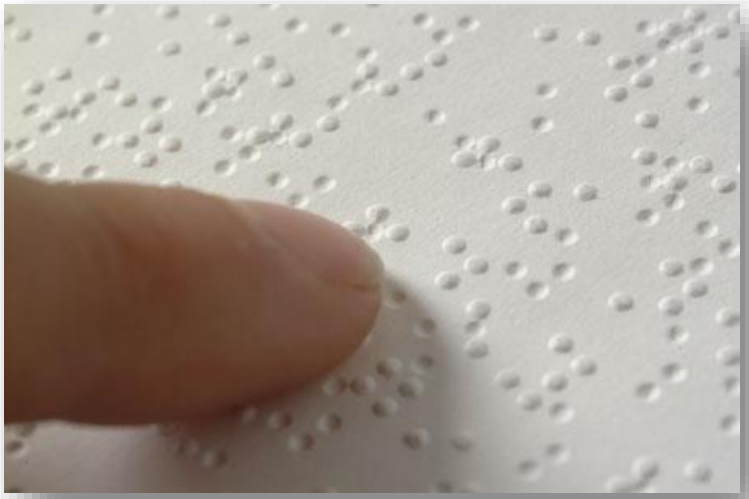
$$(110110.1)_2 = 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1}$$
$$= 32 + 16 + 0 + 4 + 2 + 0 + 0.5$$
$$= 54.5$$

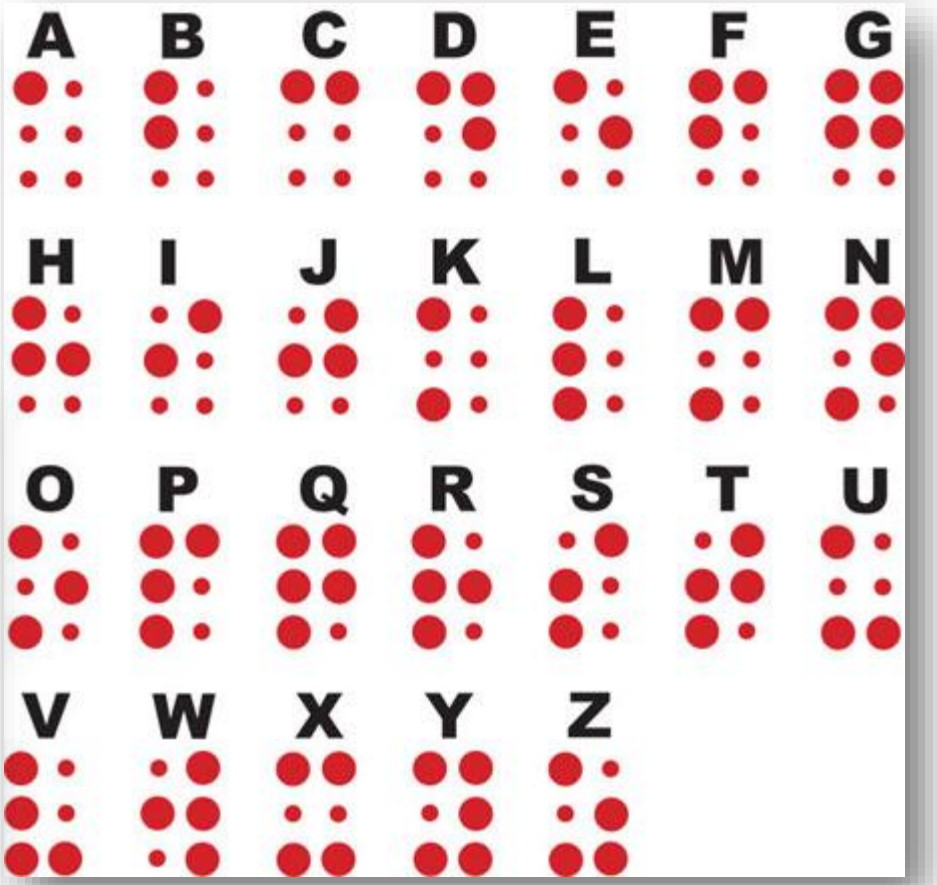
Binary-Octal-Hex

Decimal	Binary	Octal	Hex
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	3	3
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Code used by people, with low vision, to read

Braille





Special Codes

- 8-4-2-1
- BCD
- Excess-3
- Aiken
- 2-out-of-5
- …

Binary (8-4-2-1)

	8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Weighted Code

Binary Coded Decimal (BCD)

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

Excess-3

	XS3
	0000
	0001
	0010
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100
	1101
	1110
	1111

Excess-3

	XS3
	0000
	0001
	0010
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100
	1101
	1110
	1111

Symmetric code

Aiken code

	Aiken
0	0000
1	0001
2	0010
3	0011
4	0100
	0101
	0110
	0111
	1000
	1001
	1010
5	1011
6	1100
7	1101
8	1110
9	1111

2-4-2-1 Code …

Aiken code

	Aiken
0	0000
1	0001
2	0010
3	0011
4	0100
	0101
	0110
	0111
	1000
	1001
	1010
5	1011
6	1100
7	1101
8	1110
9	1111

Symmetric code

2-4-2-1 Code …

2-out-of-5

	2-out-of-5
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

It is used as Error Detecting Code; … natural even parity

2-out-of-5

	2-out-of-5
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

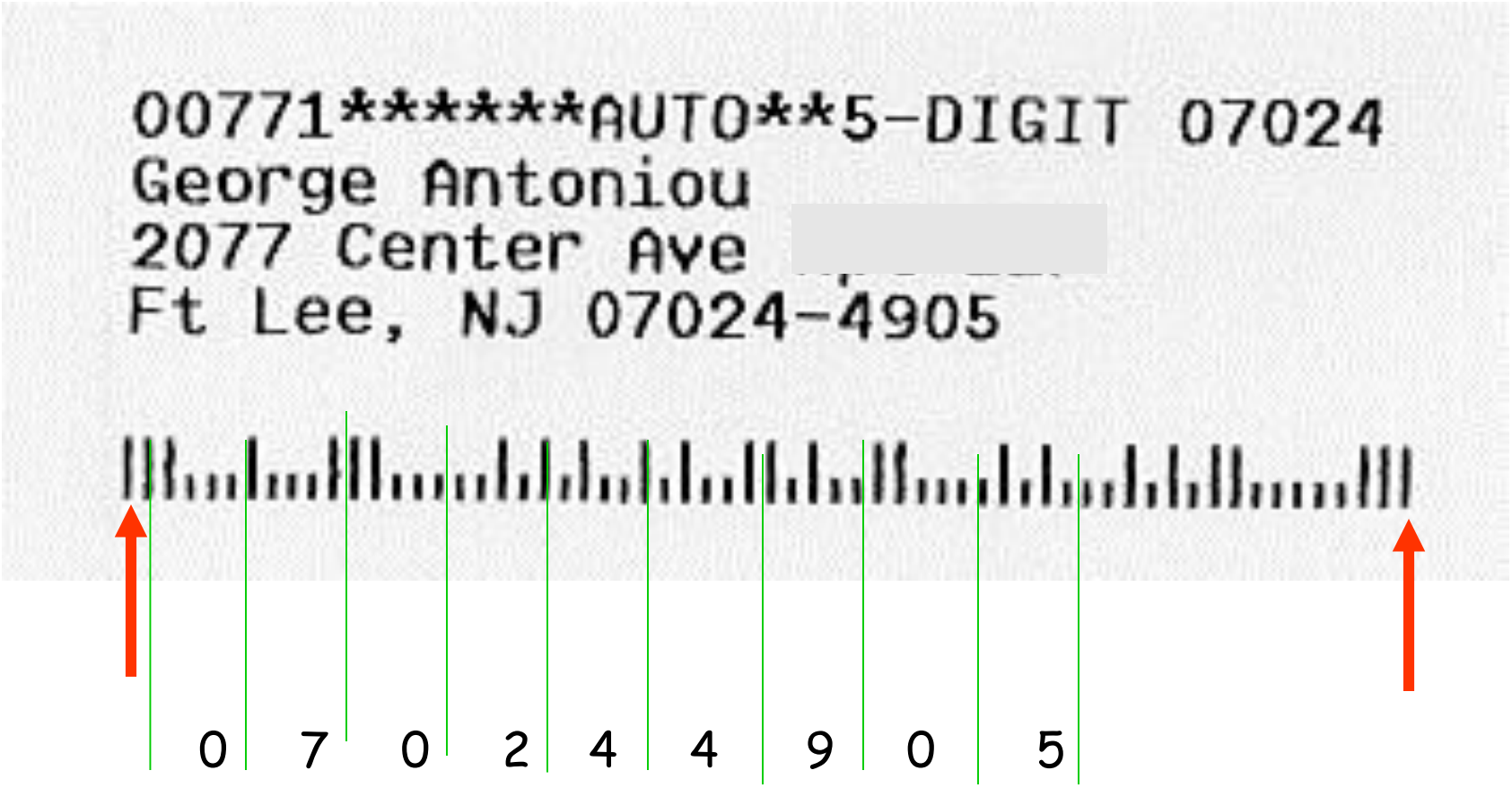
Except for the zero (0) which is decimal 24. The rest of the code follows the weights: 74210

2-out-of-5

This code is used by the U.S. Postal Service ---- Zip Code

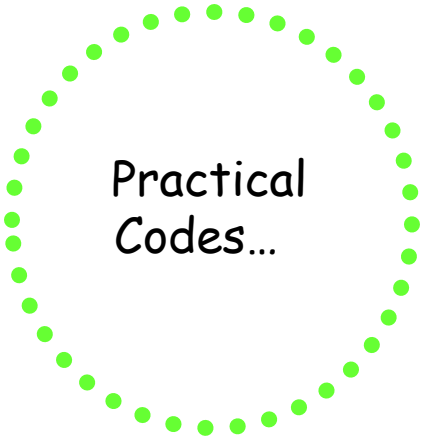
The first and last bars are the frame bars, used for aligning the scanner which reads the bar-code. The last digits are used for error correction (checksum)…..

The first and last bars are the frame bars, used for aligning the scanner which reads the bar-code. The last digits are used for error correction (checksum)…..



Bar Code based codes

- **1-D codes**
- **2-D codes**



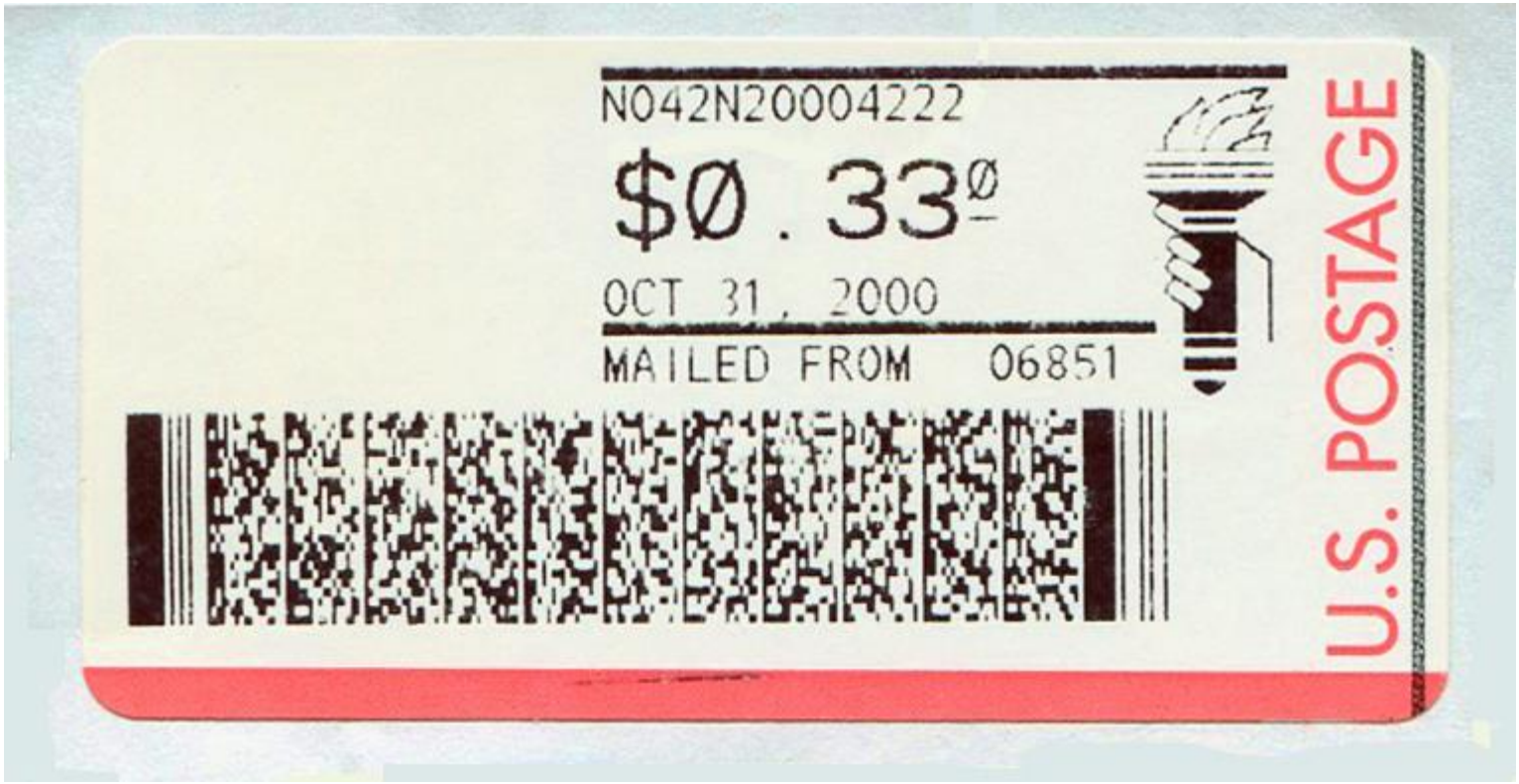
1-D and 2-D Codes **will not** be included in the exams

1-D codes

- Code 39 = Code 3-of-9; alphanumeric (full ASCII) bar code;
  applications in inventory, asset tracking, ID badges
- Interleaved 2 of 5 ; Numeric-only bar code,
  industrial applications, carton labeling, laboratory uses
- UPC (Universal Product Code), Numeric Bar code;
  used in retail product labeling.
- Code 128 – Alphanumeric (full ASCII);
  applications in Shipping, Warehouse management

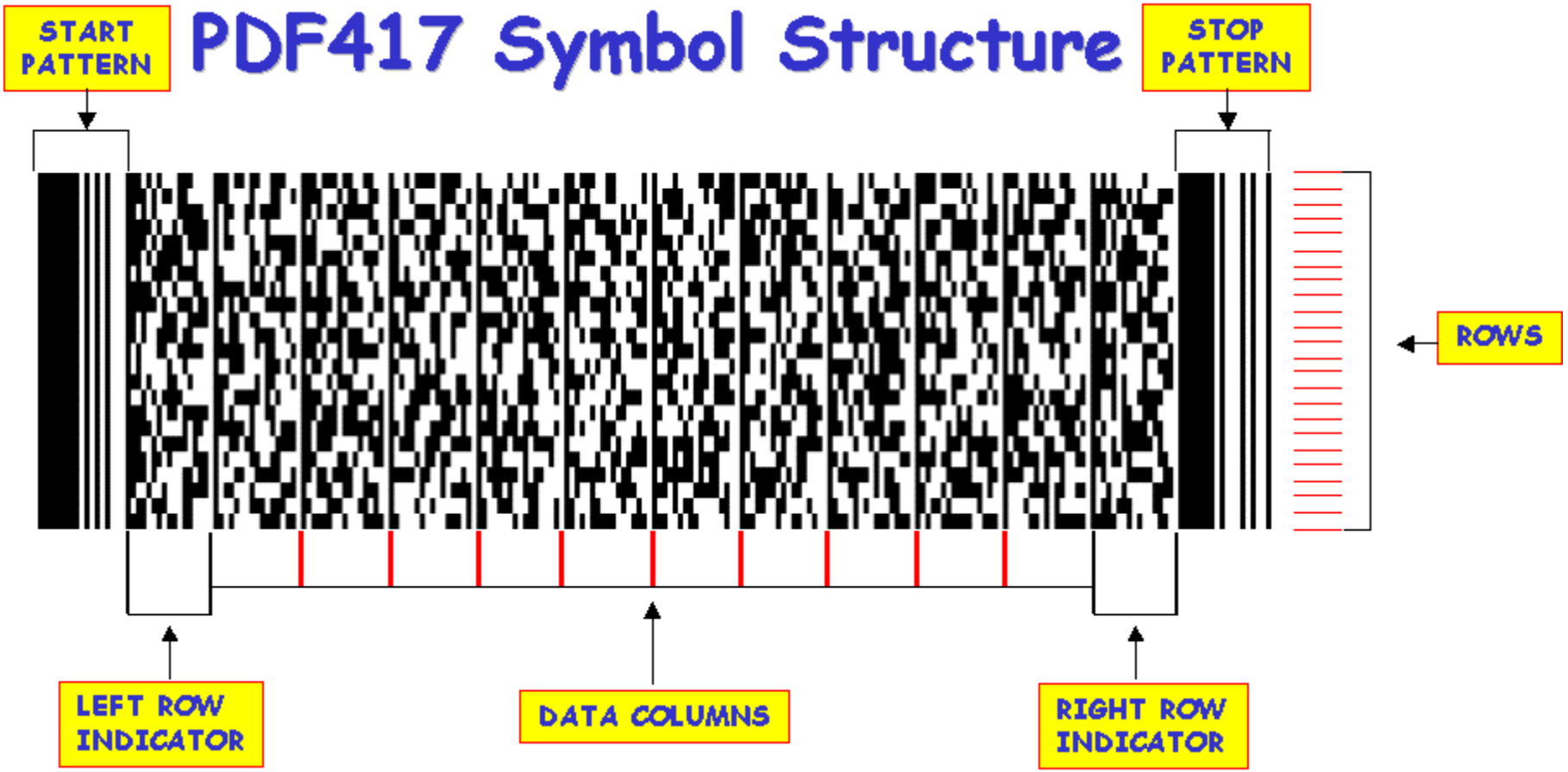
2-D Code

- -PDF417;
  applications web postage stamp……..

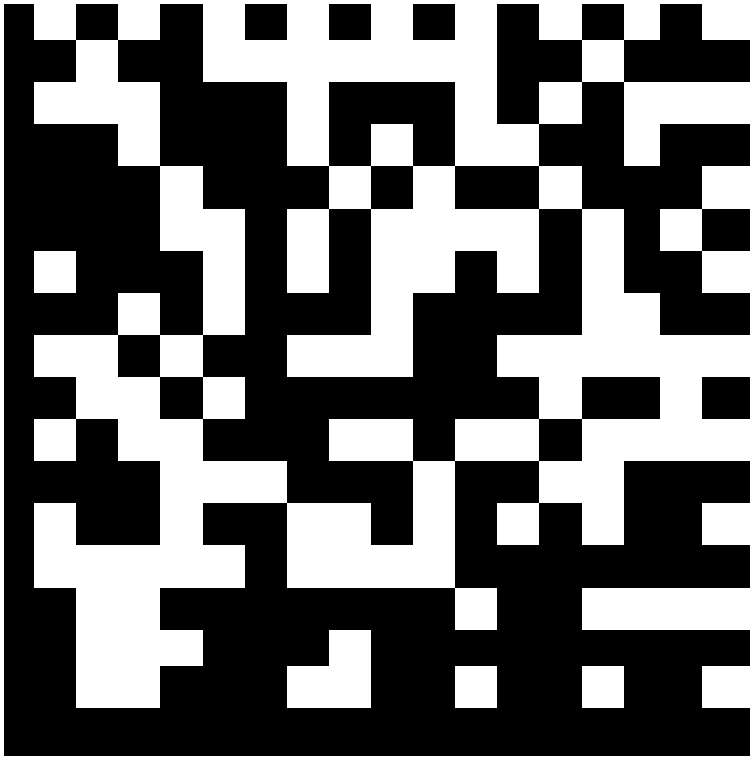


http://commons.wikimedia.org/wiki/File:USA_PC-D2.jpg

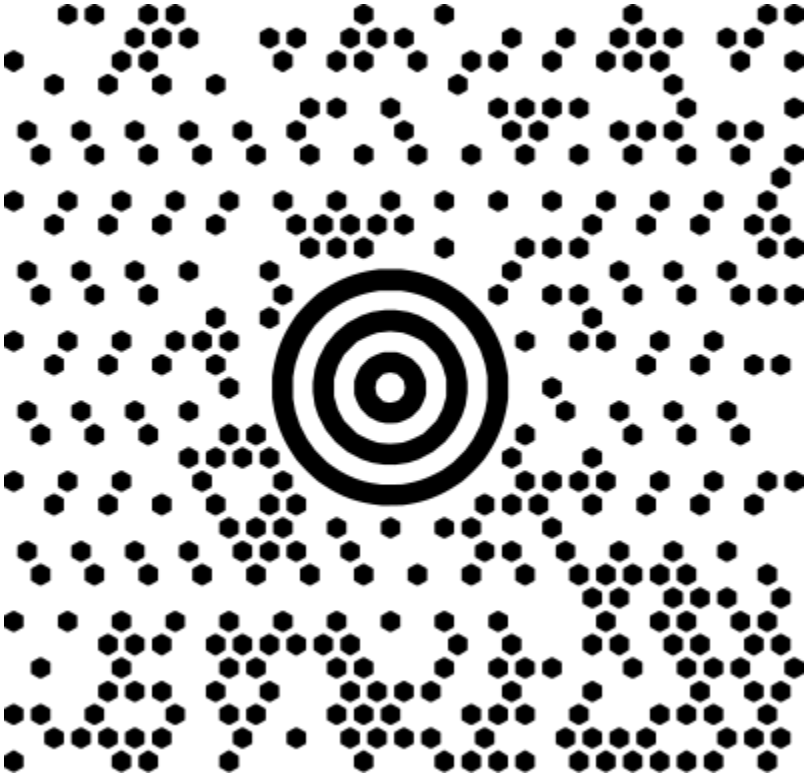
PDF417


START PATTERN
PDF417 Symbol Structure
STOP PATTERN
ROWS
LEFT ROW INDICATOR
DATA COLUMNS
RIGHT ROW INDICATOR

More 2D Codes



Data Matrix



MaxiCode

Alphanumeric codes

- Used to print, teletype or view information or other means of human alpha-numeric communication.
  - **EBCIDIC**
  - **ASCII**
  - **UniCode**

ASCII

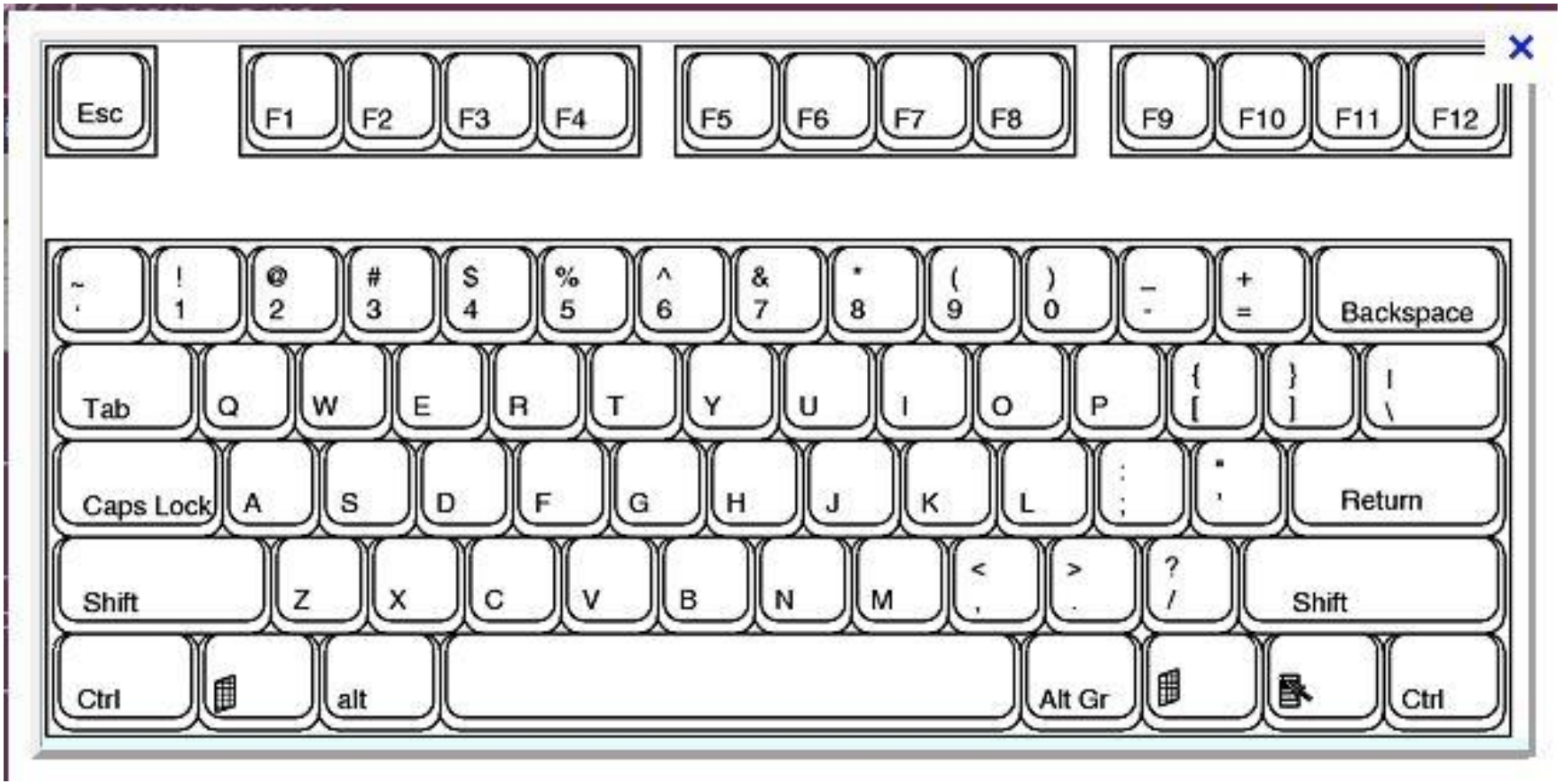
- EBCIDIC
  (Extended BCD Interchange Code)

  8 bits = 256 characters

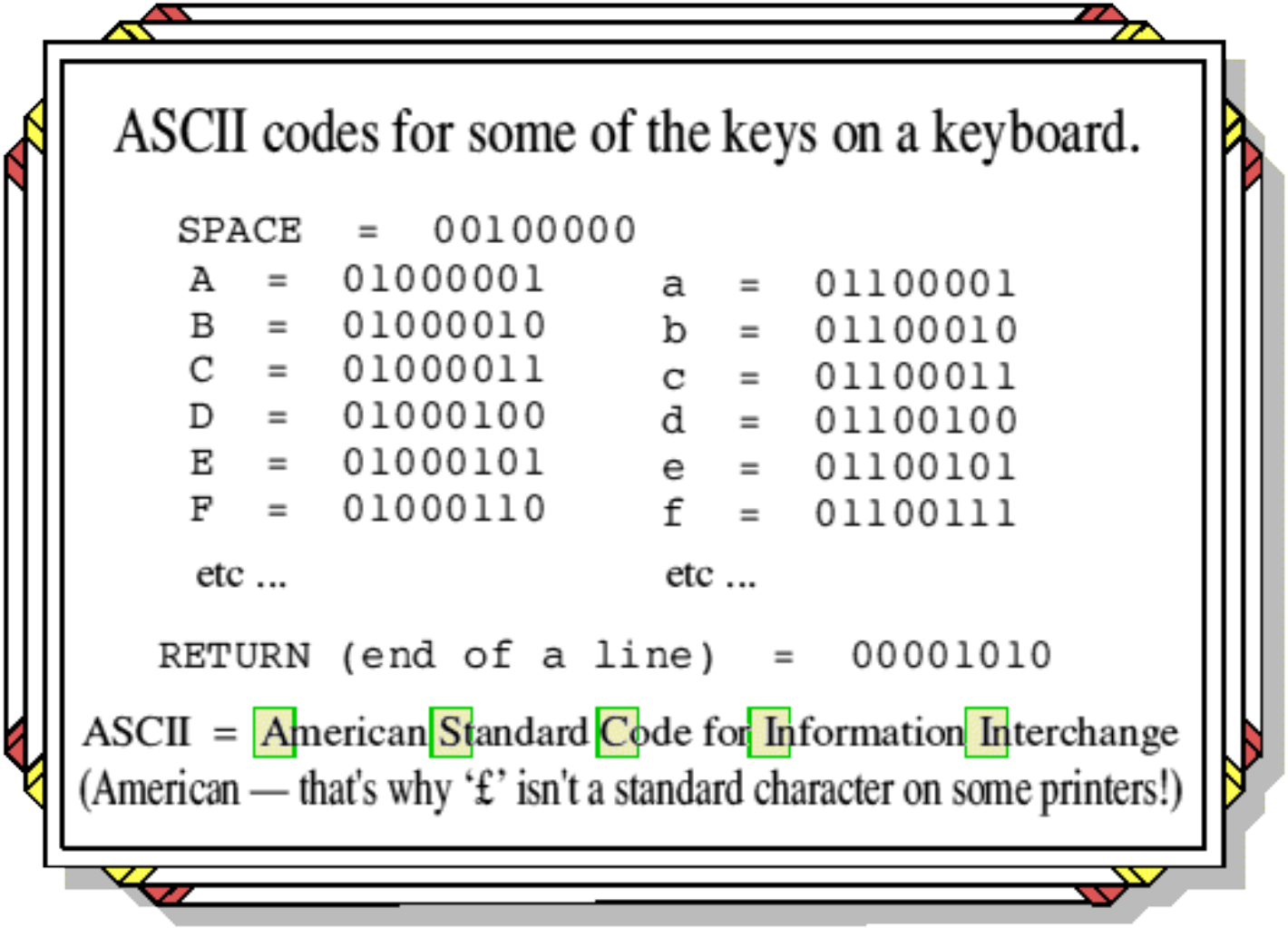
- ASCII
  (American Standard Code for Information Interchange)

  7 bits = 128 characters
  (now Super ASCII use all 8-bits)

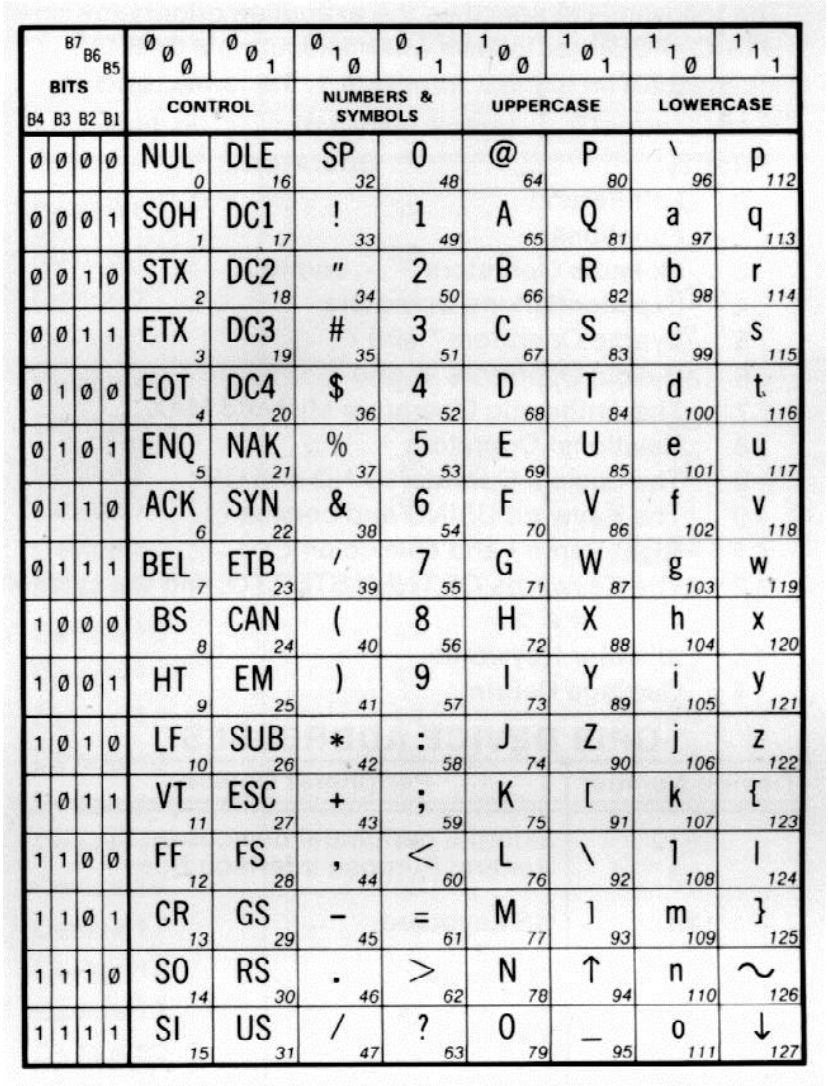
ASCII


Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
~ `
! 1
@ 2
# 3
$ 4
% 5
^ 6
& 7
* 8
(9
) 0
_ -
+ =
Backspace
Tab
Q
W
E
R
T
Y
U
I
O
P
{ [
}]
| \
Caps Lock
A
S
D
F
G
H
J
K
L
: ;
" '
Return
Shift
Z
X
C
V
B
N
M
< ,
> .
? /
Shift
Ctrl
alt
Alt Gr
Ctrl

ASCII


ASCII codes for some of the keys on a keyboard.
SPACE = 00100000
A = 01000001
B = 01000010
C = 01000011
D = 01000100
E = 01000101
F = 01000110
etc ...
a = 01100001
b = 01100010
c = 01100011
d = 01100100
e = 01100101
f = 01100111
etc ...
RETURN (end of a line) = 00001010
ASCII = American Standard Code for Information Interchange
(American — that's why '£' isn't a standard character on some printers!)

ASCII



BITS B7 B6 B5 / B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		NUMBERS & SYMBOLS		UPPERCASE		LOWERCASE	
Ø Ø Ø Ø	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
Ø Ø Ø 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
Ø Ø 1 Ø	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
Ø Ø 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
Ø 1 Ø Ø	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
Ø 1 Ø 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
Ø 1 1 Ø	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
Ø 1 1 1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1 Ø Ø Ø	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 Ø Ø 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 Ø 1 Ø	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 Ø 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 Ø Ø	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	| 124
1 1 Ø 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 Ø	SO 14	RS 30	. 46	> 62	N 78	↑ 94	n 110	~ 126
1 1 1 1	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	↓ 127

Unicode

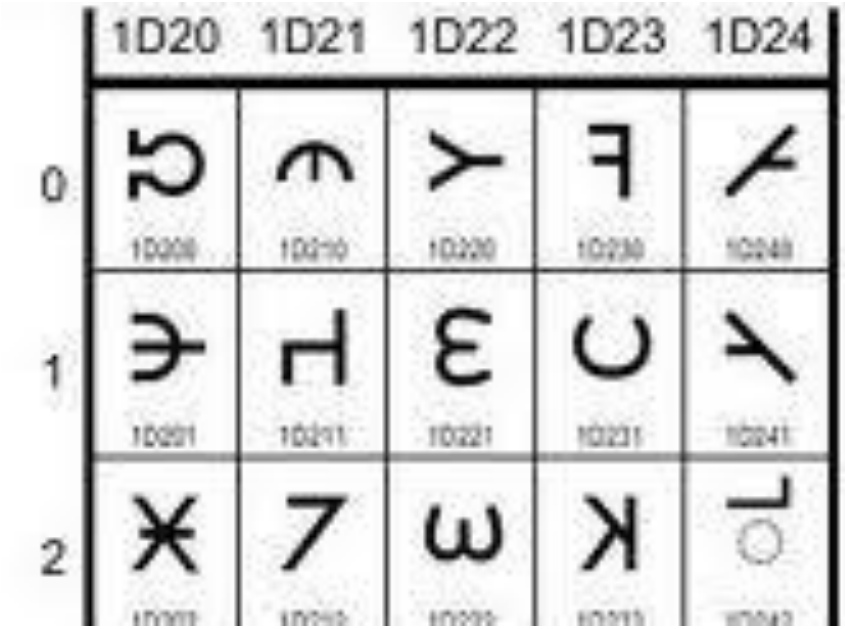
- EBCIDIC (Extended BCD Interchange Code): 8 bits = 256 characters
- ASCII (American Standard Code for Information Interchange): 7 bits = 128 characters
- Unicode (New standard for 16 – bit alphanumeric code)

UniCode

8073	8074	8075
脚 8173	腴 8174	腹 8175
艳 8273	艸 8274	艹 8275
荳 8373	茯 8374	葸 8375

Arial® Unicode® MS Bold

friður 和平 paz שלום ειρήνη 平和
mír शांति мир 평화 frieden শান্তি
WORLDWIDE LANGUAGE SUPPORT
Hòa bình ความสงบสุข Síochána



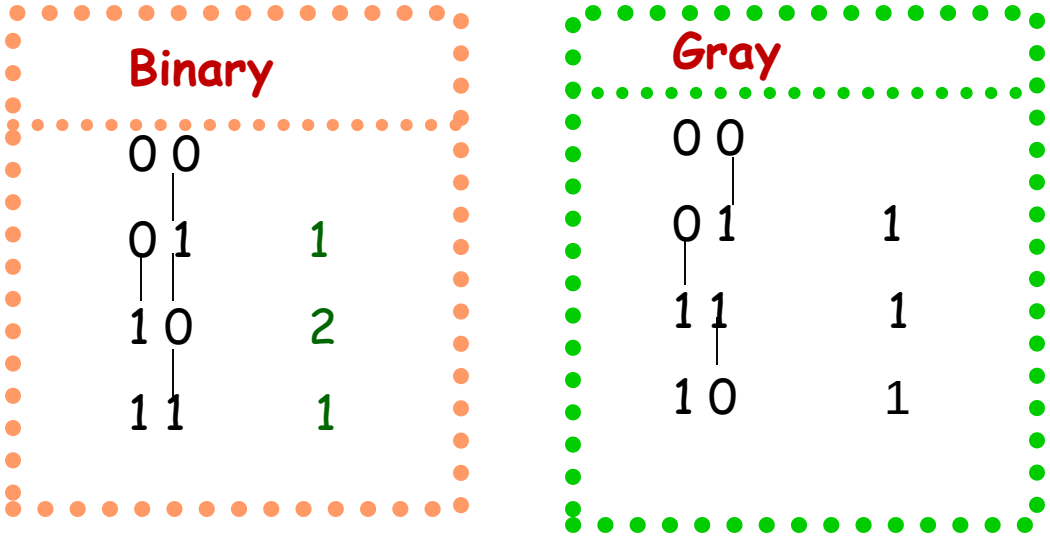
A useful code is the Gray

- Gray code is of cyclic nature having the following property:

“From one number to the next, the Gray code changes only one bit”

Bit changes …

- Gray code is of cyclic nature having the following property:
- “From one number to the next, the Gray code changes only one bit”



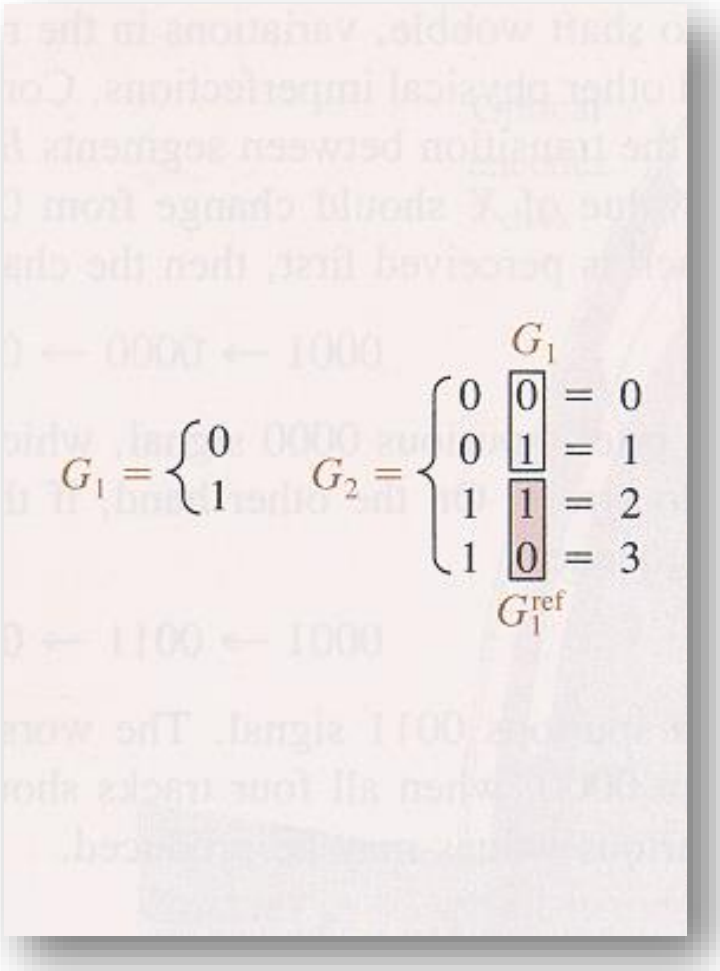
Gray Code …

- The Gray code is used for labeling **Karnaugh** maps for logic circuit simplification, for routing in parallel computer systems, etc…

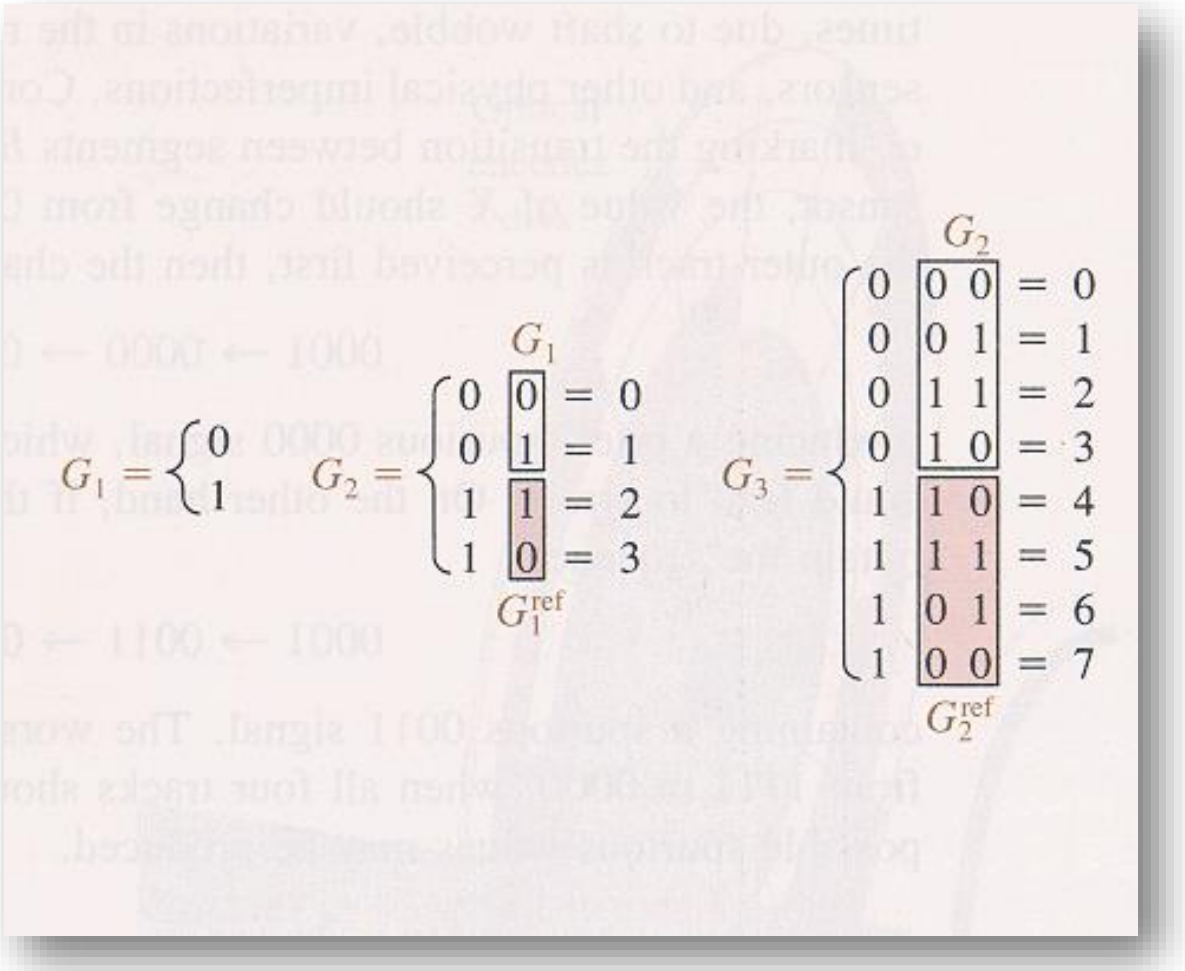
Gray Codes

$$G_1 = \begin{cases} 0 \\ 1 \end{cases}$$

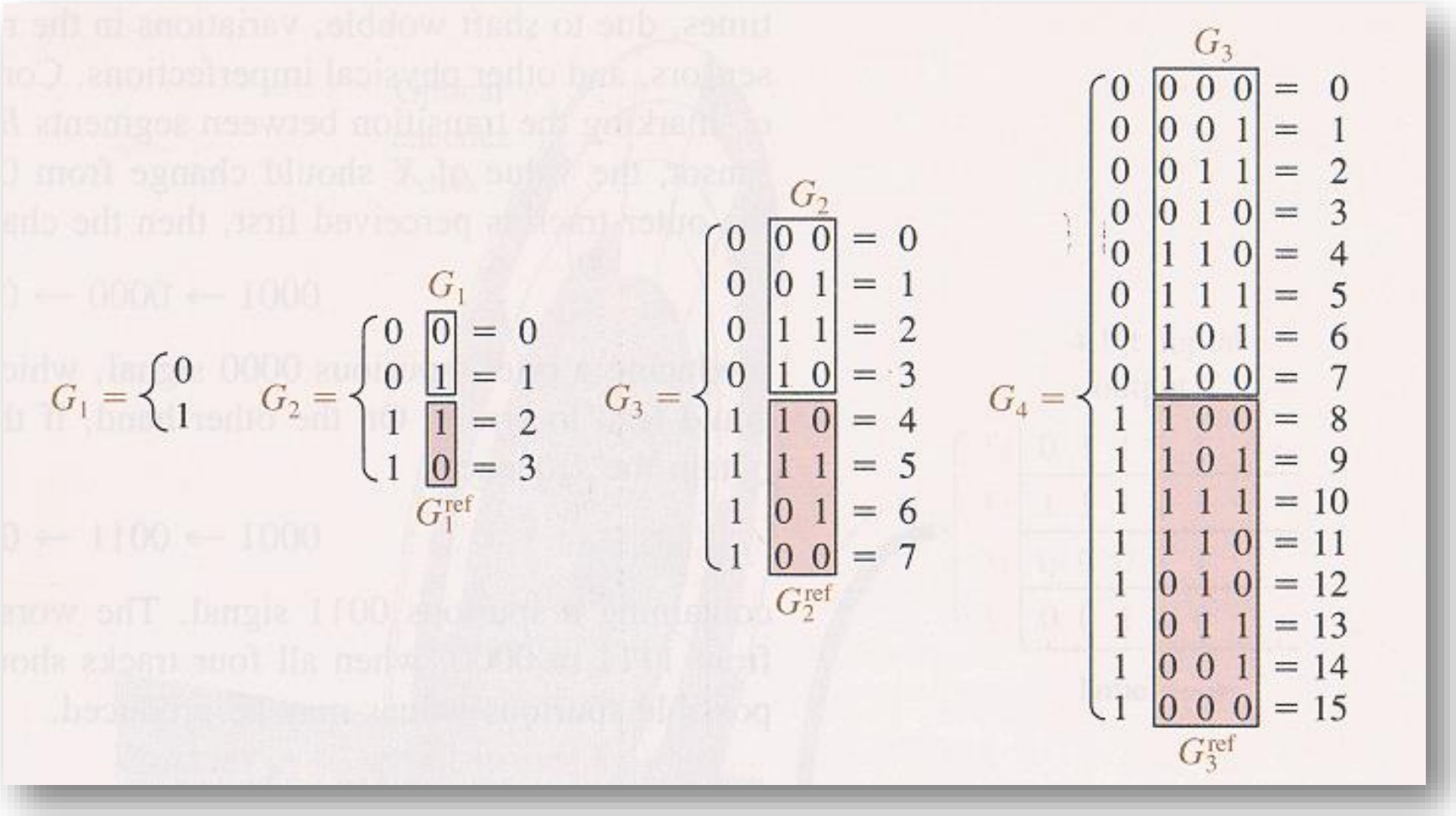
Gray Codes


$G_1 = \begin{cases} 0 \\ 1 \end{cases}$
$G_2 = \begin{cases} 0\ \boxed{0} = 0 \\ 0\ \boxed{1} = 1 \\ 1\ \boxed{1} = 2 \\ 1\ \boxed{0} = 3 \end{cases}$
G_1
G_1^{ref}

Gray Codes


$G_1 = \begin{cases} 0 \\ 1 \end{cases}$
$G_2 = \begin{cases} 0\ \boxed{0} = 0 \\ 0\ \boxed{1} = 1 \\ 1\ \boxed{1} = 2 \\ 1\ \boxed{0} = 3 \end{cases}$
G_1
G_1^{ref}
$G_3 = \begin{cases} 0\ \boxed{0\ 0} = 0 \\ 0\ \boxed{0\ 1} = 1 \\ 0\ \boxed{1\ 1} = 2 \\ 0\ \boxed{1\ 0} = 3 \\ 1\ \boxed{1\ 0} = 4 \\ 1\ \boxed{1\ 1} = 5 \\ 1\ \boxed{0\ 1} = 6 \\ 1\ \boxed{0\ 0} = 7 \end{cases}$
G_2
G_2^{ref}

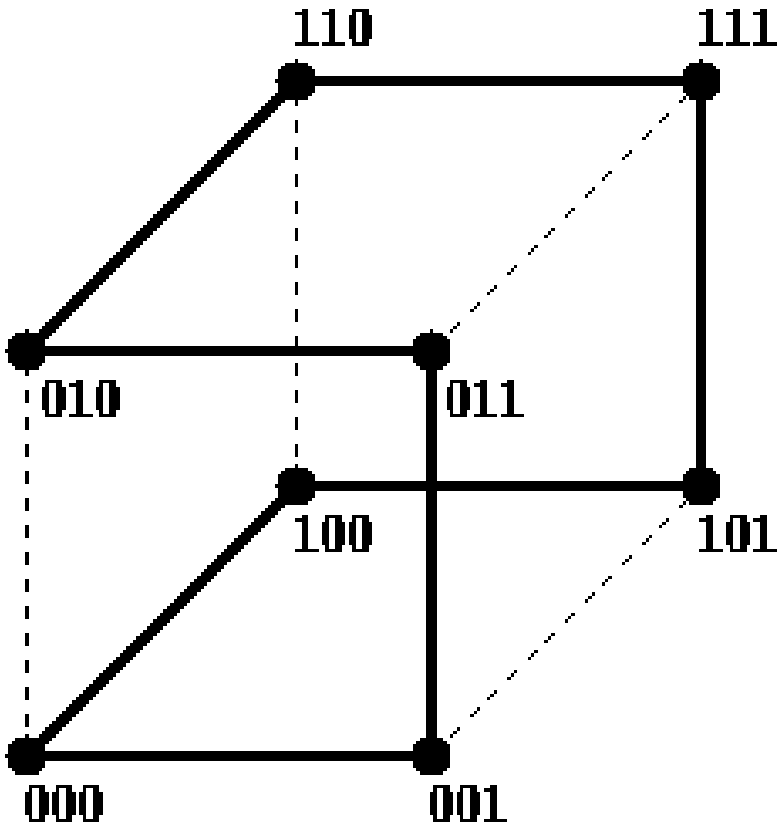
Gray Codes


$G_1 = \begin{cases} 0 \\ 1 \end{cases}$
$G_2 = \begin{cases} 0\ 0 = 0 \\ 0\ 1 = 1 \\ 1\ 1 = 2 \\ 1\ 0 = 3 \end{cases}$
G_1
G_1^{ref}
$G_3 = \begin{cases} 0\ 0\ 0 = 0 \\ 0\ 0\ 1 = 1 \\ 0\ 1\ 1 = 2 \\ 0\ 1\ 0 = 3 \\ 1\ 1\ 0 = 4 \\ 1\ 1\ 1 = 5 \\ 1\ 0\ 1 = 6 \\ 1\ 0\ 0 = 7 \end{cases}$
G_2
G_2^{ref}
$G_4 = \begin{cases} 0\ 0\ 0\ 0 = 0 \\ 0\ 0\ 0\ 1 = 1 \\ 0\ 0\ 1\ 1 = 2 \\ 0\ 0\ 1\ 0 = 3 \\ 0\ 1\ 1\ 0 = 4 \\ 0\ 1\ 1\ 1 = 5 \\ 0\ 1\ 0\ 1 = 6 \\ 0\ 1\ 0\ 0 = 7 \\ 1\ 1\ 0\ 0 = 8 \\ 1\ 1\ 0\ 1 = 9 \\ 1\ 1\ 1\ 1 = 10 \\ 1\ 1\ 1\ 0 = 11 \\ 1\ 0\ 1\ 0 = 12 \\ 1\ 0\ 1\ 1 = 13 \\ 1\ 0\ 0\ 1 = 14 \\ 1\ 0\ 0\ 0 = 15 \end{cases}$
G_3
G_3^{ref}

A Ring embedded in a 3D Hypercube

$$G_3 = \begin{cases} 0 & \boxed{0\ 0} = 0 \\ 0 & \boxed{0\ 1} = 1 \\ 0 & \boxed{1\ 1} = 2 \\ 0 & \boxed{1\ 0} = 3 \\ 1 & \boxed{1\ 0} = 4 \\ 1 & \boxed{1\ 1} = 5 \\ 1 & \boxed{0\ 1} = 6 \\ 1 & \boxed{0\ 0} = 7 \end{cases}$$

G_2 (top) and G_2^{ref} (bottom)



Binary and BCD numbers

- Bits obtained from conversion, of binary numbers, are binary digits
- Bits obtained from coding (BCD) are combinations of 1’s and 0’s arranged according to the rules of the code (BCD).

Example-1

- $(13)_{10} = (1101)_2$

Example-1

- $(13)_{10} = (1101)_2$

but,

$(13)_{10} = (0001\ \ 0011)_{BCD}$

$(1\ \ \ \ \ \ 3)_{10}$

Example-2

- $(7)_{10} = (111)_2$

Example-2

- $(7)_{10} = (111)_2$

but,

$(0111)_{BCD}$

7_{10}

Another Example

- $(15)_{10} = (1111)_2$

Another Example

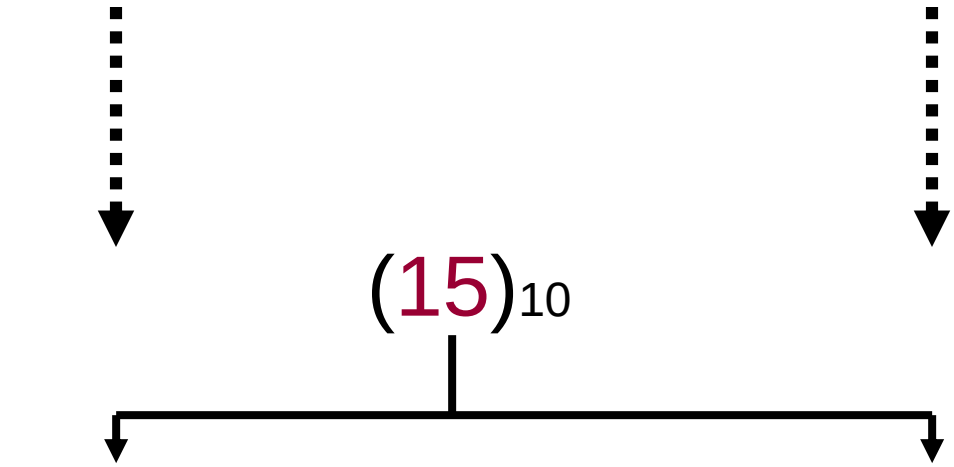
- $(15)_{10} = (1111)_2$

but,

$(15)_{10} = (0001\ \ 0101)_{BCD}$

$(1\qquad 5)_{10}$

Binary & BCD



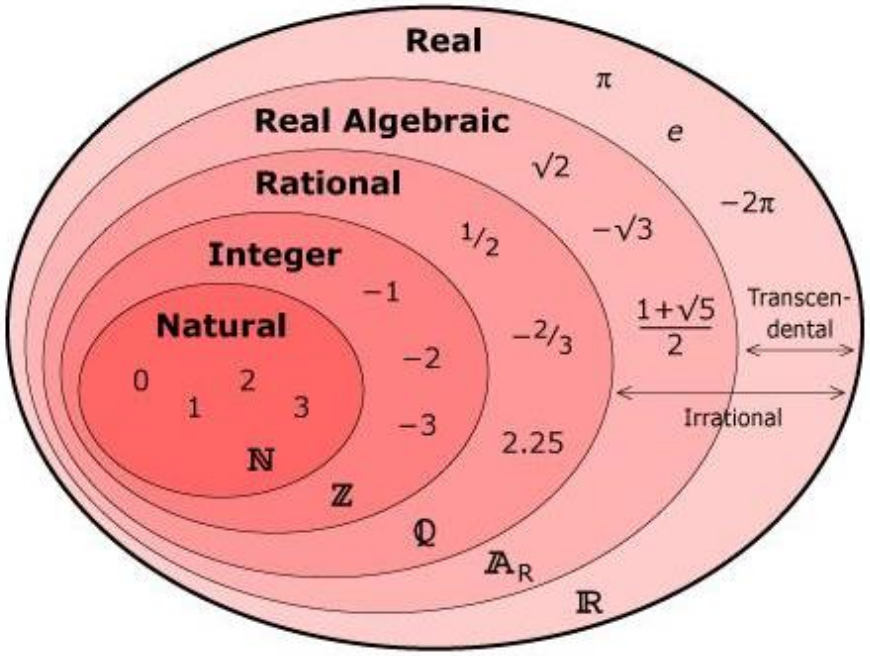
$(1111)_2$ $(0001\ \ 0101)_{BCD}$

Binary number representation

1. Representing **unsigned** integers in binary
2. Representing **signed** integers in binary
   a) Sign-magnitude
   b) Ones' complement signed magnitude
   c) Two's complement signed magnitude
3. Representing fractions in binary
   a) Fixed-point numbers
   b) Floating-point numbers

1. Representing unsigned integers

- Integers: numbers with no fractional part
  - Zero
  - Positive
  - Negative
- Integers … straight binary representation



Unsigned binary

- The range is: 0 up to $2^n - 1$ (min – max)

- Example-1: n = 3: 0 → 8-1 = 7
- Example-2: n = 8: 0 → 256-1 = 255

3-bit unsigned number

Decimal	Unsigned
7	111
6	110
5	100
4	100
3	011
2	010
1	001
0	000

8-bit unsigned numbers

Binary	Unsigned
00000000	0
00000001	1
…	…
01111111	127
10000000	128
10000001	129
…	…
11111111	255

$2^8 - 1 = 255$

Largest unsigned 32-bit

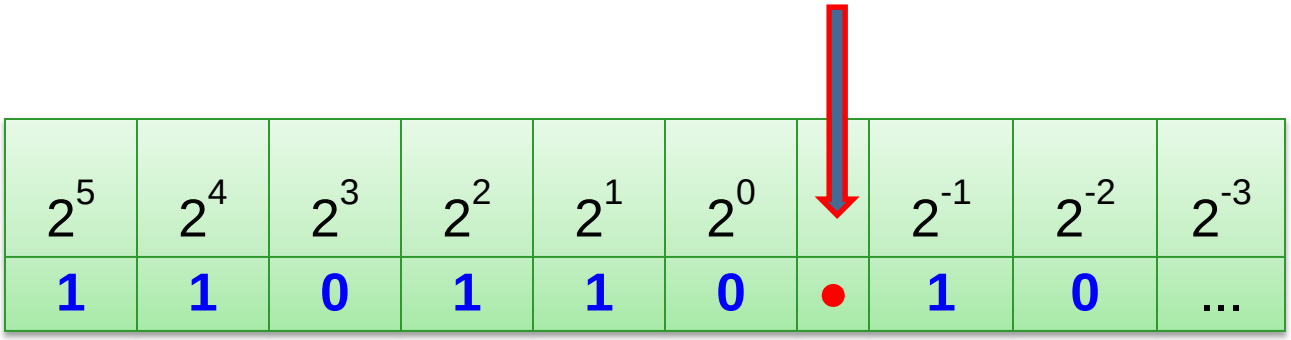
- For a 32-bit computer …
- What is the largest unsigned integer that can hold?

11111111-11111111-11111111-11111111

$2^{32}-1 = 4,294,967,295$

Binary unsigned numbers

$(54.5)_{10} = (110110.1)_2$



$(110110.1)_2 = 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1}$

$= 32 + 16 + 0 + 4 + 2 + 0 + 0.5$

$= 54.5$

Negative unsigned binary number?

- Need to place a minus sign in front of the number … correct?

- Binary number: -100101

- This is not a binary number system … it is a trinary !
- Use **signed binary** numbers

2. Representing signed integers in binary

a) Signed-magnitude
b) 1’s complement signed magnitude
c) 2’s complement signed magnitude

a) Signed-magnitude

- A n-bit binary number has 2^n distinct values.
  - assign about **half to positive integers** and about **half to negative**
  - that leaves two values: **one for the 0**, and **one extra**
- Positive integers
  - just like unsigned – **zero** in *most significant* bit
- Negative integers
  - Sign-magnitude – **one** in most significant bit

Most significant bit: **0 = positive**, **1 = negative**

Negative-Positive

- Positive and Negative numbers:
  - 1 in the MSB position is negative
  - 0 in the MSB position is positive
- Example:

1 000-0010	-2
0 000-0010	+2

3-bit unsigned-magnitude numbers

Decimal	Unsigned
+7	111
+6	110
+5	100
+4	100
+3	011
+2	010
+1	001
0	000

3-bit signed-magnitude numbers

- A 3-bit binary number has a total of 2^3 = 8 numbers.
- Using signed number representation, we have negative and positive numbers: Divide 8 by 2 → 4. We have:
  - 4-positive (+0 → +3)
  - 4-negative (-0 → -3) numbers.
- The only «problem» is that we have 2 different zeros (-0, +0).

The range is: +3 → -3
We have 2 ways to represent zero

Decimal	Unsigned	Signed
+7	111	
+6	110	
+5	100	
+4	100	
+3	011	011
+2	010	010
+1	001	001
+0	000	000
-0		100
-1		101
-2		110
-3		111
-4		

Range: Signed-magnitude numbers

The range is: +3 → -3

In general

$2^{n-1}-1$ to $-(2^{n-1}-1)$

Decimal	Unsigned	Signed
+7	111	
+6	110	
+5	100	
+4	100	
+3	011	011
+2	010	010
+1	001	001
+0	000	000
-0		100
-1		101
-2		110
-3		111
-4		

For n = 4

B	Value represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	- 0	- 7	- 8
1 0 0 1	- 1	- 6	- 7
1 0 1 0	- 2	- 5	- 6
1 0 1 1	- 3	- 4	- 5
1 1 0 0	- 4	- 3	- 4
1 1 0 1	- 5	- 2	- 3
1 1 1 0	- 6	- 1	- 2
1 1 1 1	- 7	- 0	- 1

8-bit signed-magnitude

Binary	Signed	Unsigned
00000000	+0	0
00000001	1	1
...	...	...
01111111	127	127
10000000	−0	128
10000001	−1	129
...	...	...
11111111	−127	255

Not used in modern computing … because there are two representations for the zero

IBM7090/ 1959

Range of n-bit numbers

bits	Binary			
	Unsigned		Sign-magnitude	
	Min	Max	Min	Max
n	0	$2^n - 1$	$-(2^{n-1} - 1)$	$2^{n-1} - 1$

1’s complement-binary numbers

- Replace … all zeros with ones, and ones with zeros

- 100001
- 1’s complement: 011110

b) 3-bit 1's complement signed-magnitude

The range is: +3 → -3

In general

$2^{n-1}-1$ to $-(2^{n-1}-1)$

We have 2 ways to represent zero

Decimal	Unsigned	Signed	1's Comp.
+7	111		
+6	110		
+5	100		
+4	100		
+3	011	011	011
+2	010	010	010
+1	001	001	001
→ +0	000	000	000
→ -0		100	111
-1		101	110
-2		110	101
-3		111	100
-4			

8-bit 1’s complement signed-magnitude

Binary value	1' complement	Unsigned
00000000	+0	0
00000001	1	1
...	...	...
01111101	125	125
01111110	126	126
01111111	127	127
10000000	−127	128
10000001	−126	129
10000010	−125	130
...	...	...
11111110	−1	254
11111111	−0	255

Not used in modern computing … because there are two representations for the zero

CDC160A/1960
UNIVAC 1100/2200 /1962

2’s complement-binary numbers

- Having the binary number in 1’s complement …
- Add 1 …
- … the result is the 2’s complement of the initial binary number

- Binary number: 100001
- 2’s complement: 011110 + 1 = 011111

c) 3-bit 2's complement signed magnitude

Decimal	Unsigned	Signed	1's Comp.	2's Comp
+7	111			
+6	110			
+5	100			
+4	100			
+3	011	011	011	011
+2	010	010	010	010
+1	001	001	001	001
+0	000	000	000	000
-0		100	111	000
-1		101	110	111
-2		110	101	110
-3		111	100	101
-4				100

The range is:
Positive (+0 → +3)
Negative (-1 → -4)
… we have 1 way to represent zero

8-bit 2’s comp-signed magnitude

Binary value	2’s complement	Unsigned
00000000	0	0
00000001	1	1
…	…	…
01111110	126	126
01111111	127	127
10000000	−128	128
10000001	−127	129
10000010	−126	130
…	…	…
11111110	−2	254
11111111	−1	255

2’s complement signed-magnitude

- Today’s processors represent signed integers using two’s complement … Why? … because…
- A 2’s complement signed-magnitude representation has a single representation for zero (0)

Range for n-bit 2's complement signed

Range: $-2^{n-1} \rightarrow 2^{n-1} - 1$

For our 3-bit [$2^3 = 8$ … divide by 2 ➔ $4 = 2^2$]
For our example the range is (-4 → +3):

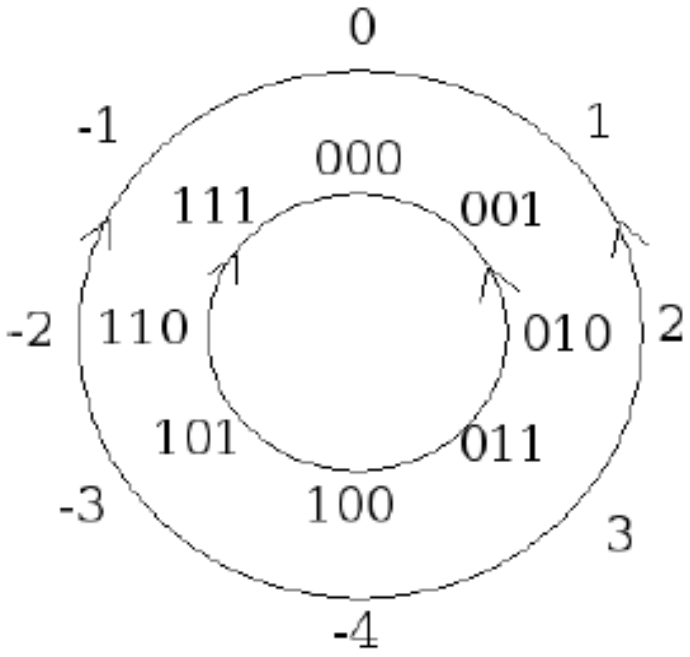
-

$2^{3-1} \rightarrow 2^{3-1} - 1 = -2^2 \rightarrow 2^2 - 1 = -4 \rightarrow +3$

Modulo-16 system (4-bits)

B	Value represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0 ←	+ 0 ←	+ 0 ←
1 0 0 0	- 0 ←	- 7	- 8
1 0 0 1	- 1	- 6	- 7
1 0 1 0	- 2	- 5	- 6
1 0 1 1	- 3	- 4	- 5
1 1 0 0	- 4	- 3	- 4
1 1 0 1	- 5	- 2	- 3
1 1 1 0	- 6	- 1	- 2
1 1 1 1	- 7	- 0 ←	- 1

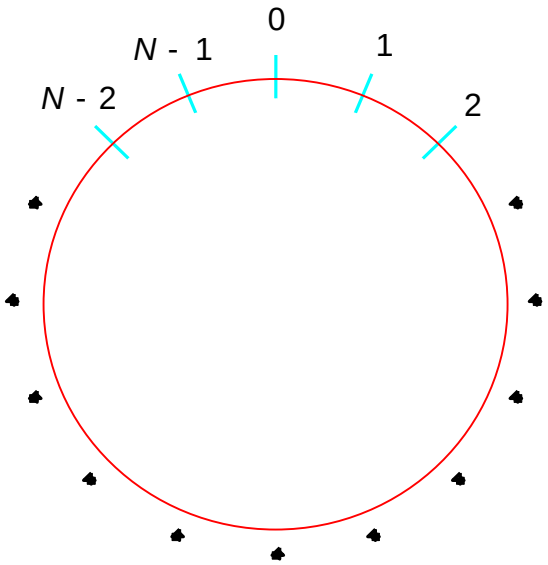
3-bit signed 2’s complement representation



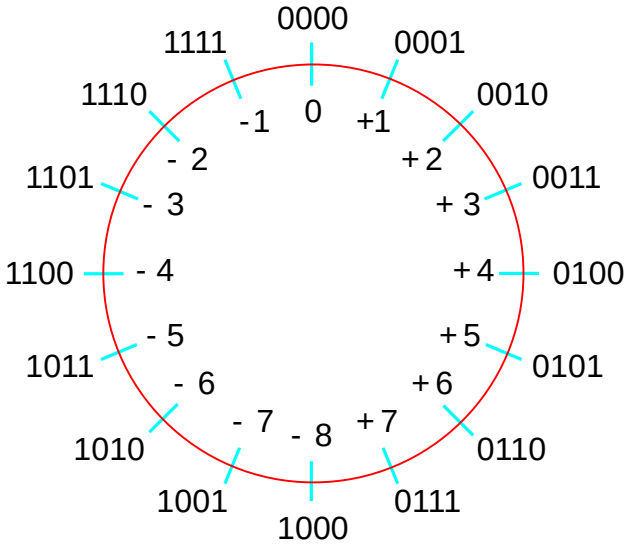
2s complement

- 2's complement numbers actually make sense since they follow normal modulo arithmetic except when they overflow

- Range is -2^{n-1} to $2^{n-1}-1$



(a) Circle representation of integers mod *N*



(b) Mod 16 system for 2's-complement numbers

Range of n-bit numbers

bits	Binary					
	Unsigned		Sign/Magnitude		2's Complement	
	Min	Max	Min	Max	Min	Max
n	0	$2^n - 1$	$-(2^{n-1} - 1)$	$2^{n-1} - 1$	-2^{n-1}	$2^{n-1} - 1$

2’s complement signed → decimal

2's complement signed → decimal

Signed 2's Complement

Decimal

$$a_{n-1}a_{n-2}\ldots a_1a_0 \quad = \quad -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

for $n = 3$, yields

$n = 3$, 2's complement signed → decimal

$$a_{n-1}a_{n-2}\ldots a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}\ldots a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$n = 3$, 2's complement signed → decimal

$$a_{n-1}a_{n-2}...a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}...a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

Example: $100_{SM\ 2C}$ to decimal

$$a_{n-1}a_{n-2}...a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}...a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + 2^0 \times a_0 + 2^1 \times a_1$$

Example: $100_{SM\ 2C}$ to decimal

$$a_{n-1}a_{n-2}...a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}...a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + 2^0 \times a_0 + 2^1 \times a_1$$

$$100 = -2^2 \times 1 + 2^0 \times 0 + 2^1 \times 0$$

Example: $100_{SM\ 2C}$ to decimal

$$a_{n-1}a_{n-2}...a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}...a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + 2^0 \times a_0 + 2^1 \times a_1$$

$$100 = -2^2 \times 1 + 2^0 \times 0 + 2^1 \times 0$$

$$100 = -4 + 0 + 0$$

Example: $100_{SM\ 2C}$ to decimal

$$a_{n-1}a_{n-2}...a_1a_0 = -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

$$a_{3-1}a_{3-2}...a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + \sum_{i=0}^{1} 2^i \times a_i$$

$$a_2a_1a_0 = -2^2 \times a_2 + 2^0 \times a_0 + 2^1 \times a_1$$

$$100 = -2^2 \times 1 + 2^0 \times 0 + 2^1 \times 0$$

$$100 = -4 + 0 + 0$$

$$100 = -4$$

$100_{SM2C} = -4$

Decimal	Unsigned	Signed	1's Comp.	2's Comp
+7	111			
+6	110			
+5	100			
+4	100			
+3	011	011	011	011
+2	010	010	010	010
+1	001	001	001	001
+0	000	000	000	000
-0		100	111	000
-1		101	110	111
-2		110	101	110
-3		111	100	101
-4				100

Position weighting with sign-bit for n = 3

Bit	a_2	a_1	a_0
weight	-4	2	1

Range of 32-bit 2's comp. signed numbers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$

…

$0111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111_2 = 2^{31}-1$

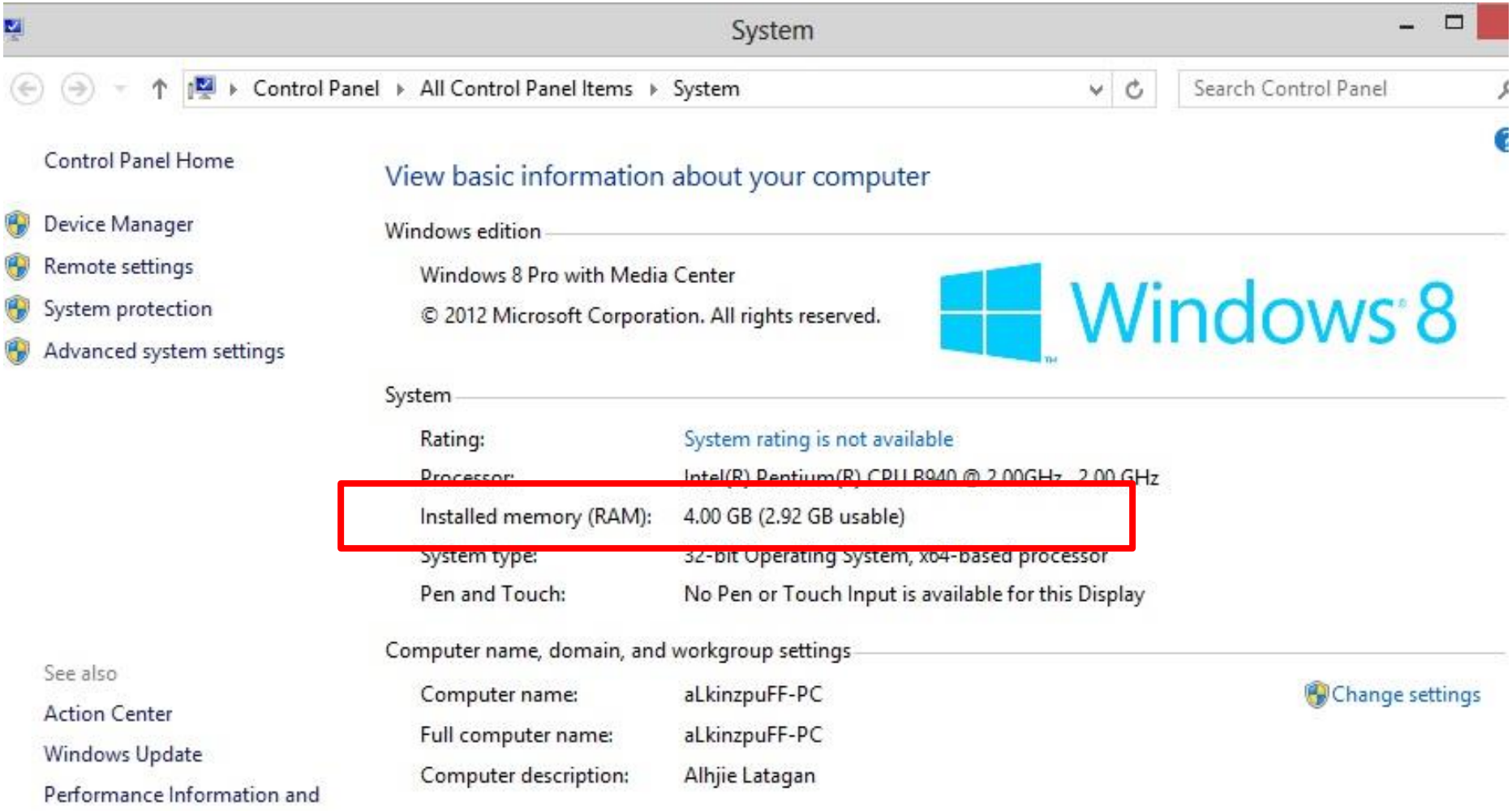
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2^{31}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -(2^{31} - 1)$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -(2^{31} - 2)$

$2^{31}-1 = 4{,}294{,}967{,}295$

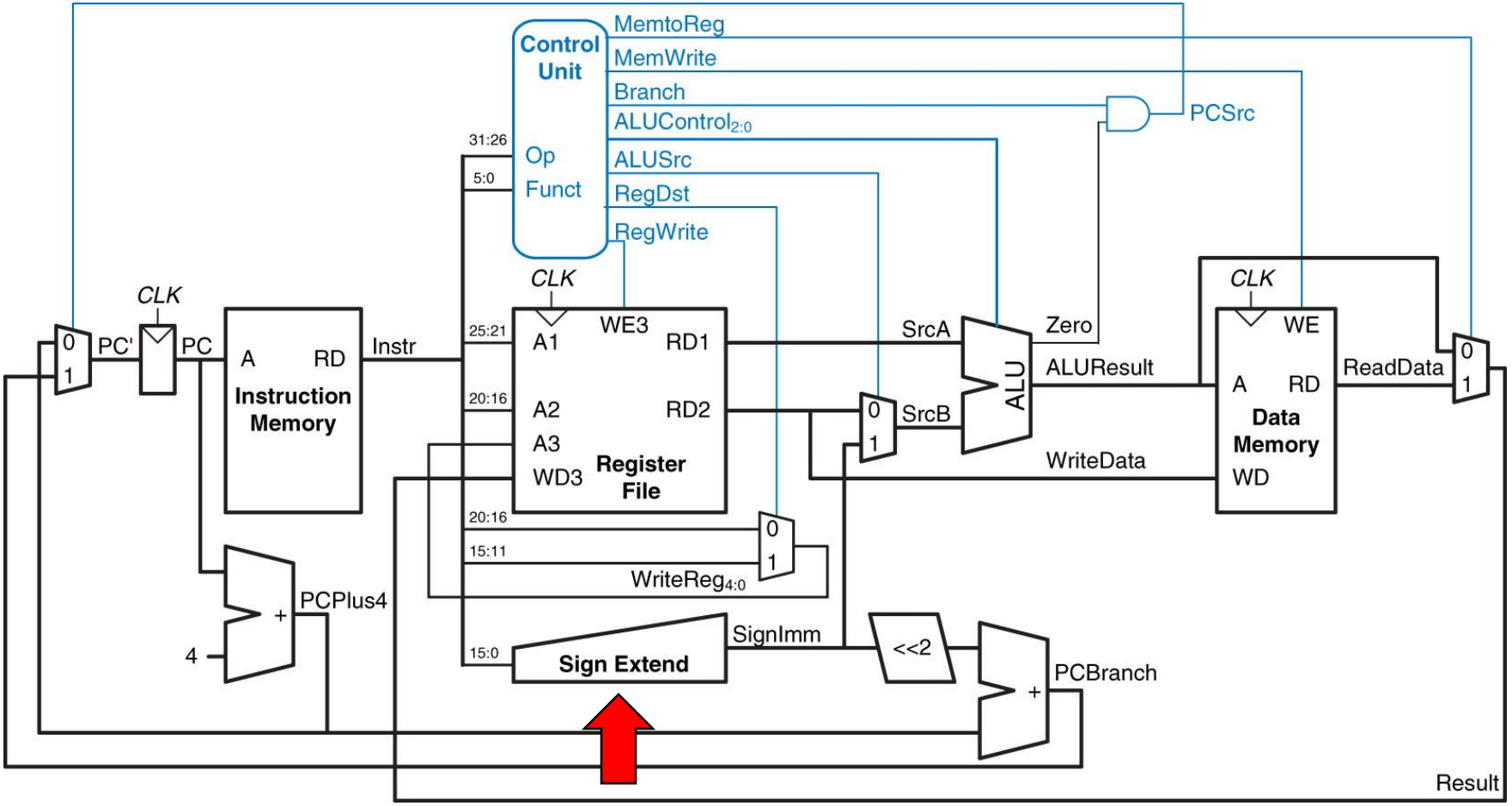
32 bit OS - 4 GB RAM



Sign and Zero-Extend

- Extend number from N to M bits (M > N)
  - Sign-Extend
  - Zero-Extend

Single-Cycle MIPS Processor



MIPS instruction: Sign Extend

- Arithmetic instructions (**`add, sub, …,ori`**) sign extend **`immed`**(iates).

- Also used for unsigned arithmetic.

Sign-Extend (4-bits to 8-bits)

- Sign bit copied to Most Significant Bits
- Number value is same

❑ **Example 1:**

– 4-bit representation of 3 = 0011 ⬅

– 8-bit sign-extended value: 00000011 ⬅

❑ **Example 2:**

– 4-bit representation of -3 = 1011 ⬅

– 8-bit sign-extended value: 11111011 ⬅

Zero-Extend (4-bits to 8-bits)

- Zeros copied to Most Significant Bits
- Value changes for negative numbers

- **Example 1:**
  - 4-bit value = $0011_2 = 3_{10}$
  - 8-bit zero-extended value: 00000011 $= 3_{10}$
- **Example 2:**
  - 4-bit value = 1011 $= -3_{10}$
  - 8-bit zero-extended value: 00001011 $= 3_{10}$

From negative to positive with zero MSB

Padding with zeros on the left

Sign-Extend in MIPS (to 32-bits)

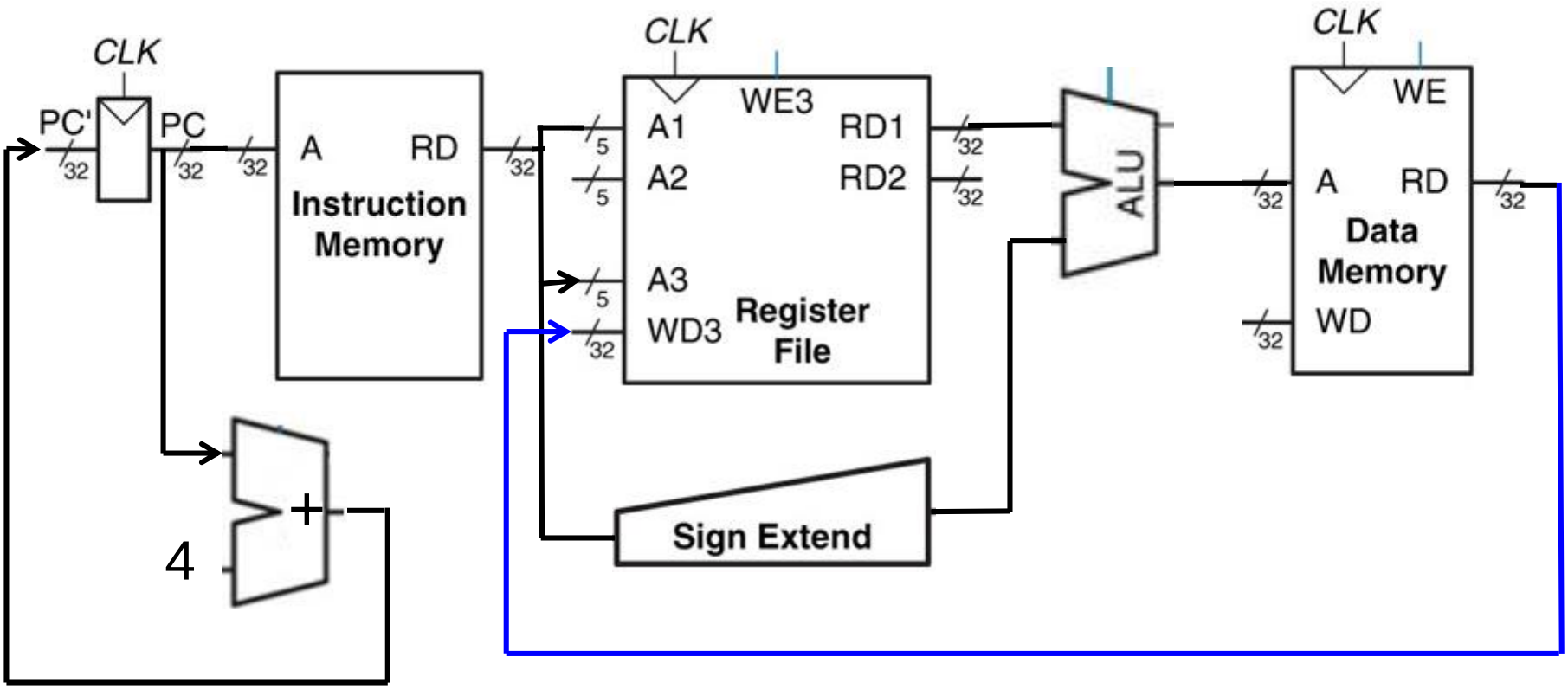
- Instructions like: `addiu $t1,$t1,1`
- Add 16-bit and 32-bit values together.
- Extend the sign to the left, so both number are 32-bits…

Zero-Extend in MIPS (to 32-bits)

Instructions like: `ori $t1,$0,0x3`

- Extend the 16-bit to 32-bits
- Fill with zero's to the left, so both numbers are 32-bits

Sign-Extend in MIPS



32-bit and 64-bit

32-bit:

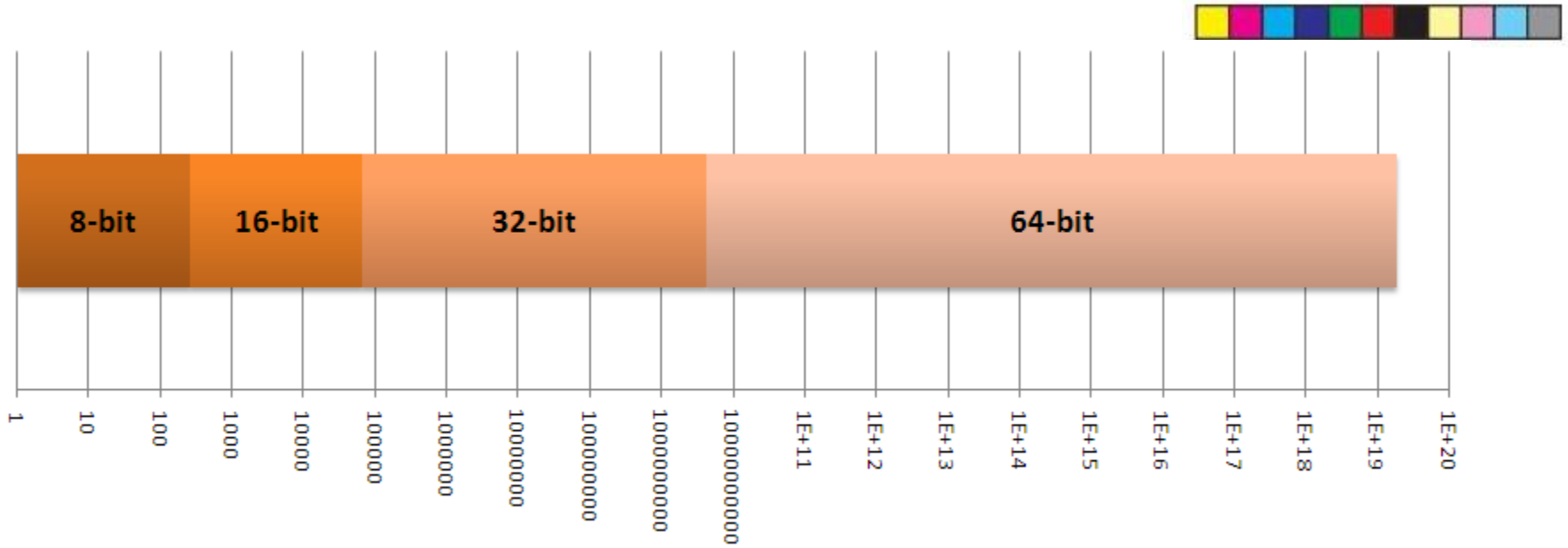
2^{32} = 4,294,967,296
4,294,967,296 / (1,024 x 1,024) = 4,096 MB = 4GB (gigabytes)

64-bit:

2^{64} = 18,446,744,073,709,551,616
18,446,744,073,709,551,616 / (1,024 x 1,024) = 16EB (exabytes)

Note that … giga >> tera >> peta >> exa

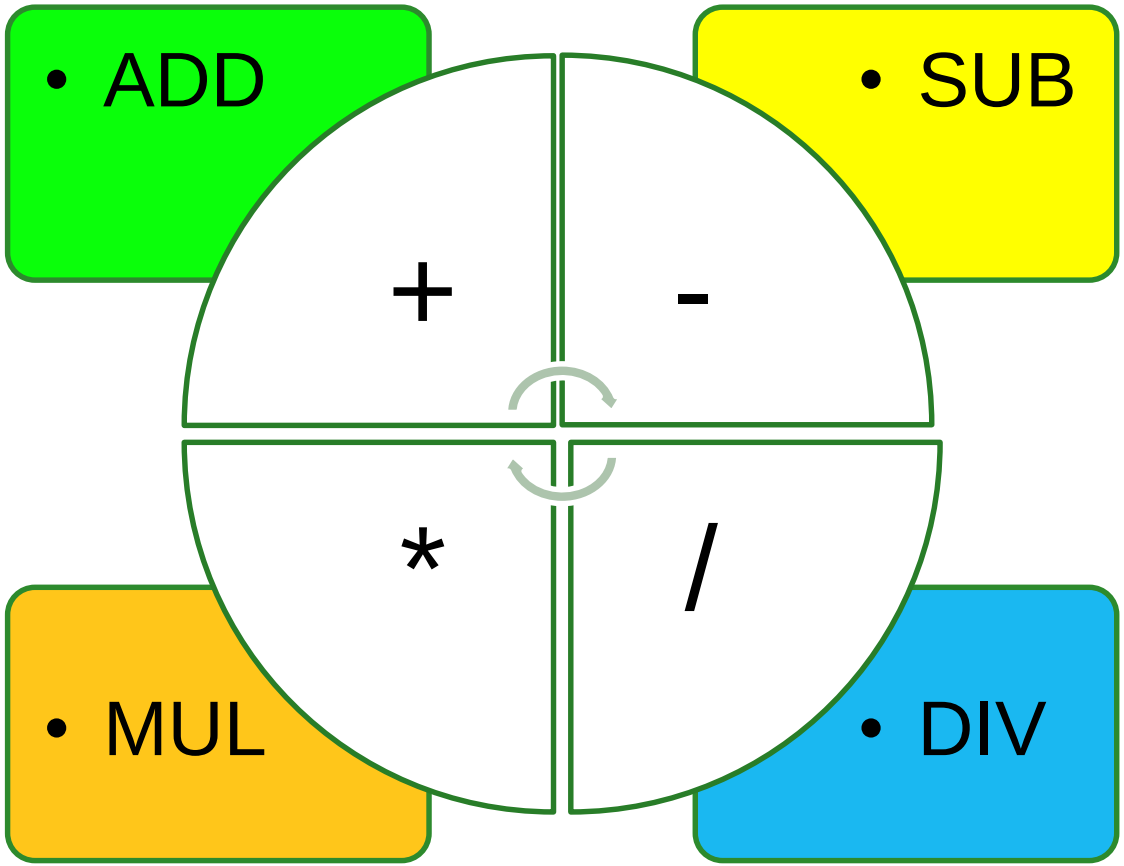
32-bit and 64-bit



Example of general purpose processor evolution: More processing ability, more addressable memory

64-bit computers can realistically access 4 GB and 128 GB of RAM.

Arithmetic Operations



ALU and Math operations are the heart of the CPU

- The computer-CPU (ALU) performs math operations ?
- Main mathematical operation in the ALU ?
- **ADD**
- All the other mathematical operations are performed, using addition, and …
- HOW ?

Addition

Decimal addition

$$\begin{array}{r} 369 \\ +\ 32 \\ \hline 401 \end{array}$$

Binary addition

1

+0

———

1 = sum

Binary addition

1	0
+0	+1
———	———
1 = sum	1 = sum

Binary addition

1	0	1
+0	+1	+1
1 = sum	1 = sum	0 = sum
		1 = carry

Binary example

10110

+10011

———————

1 ← Result

Binary example

0 1 1 0 0 ← Carry

10110

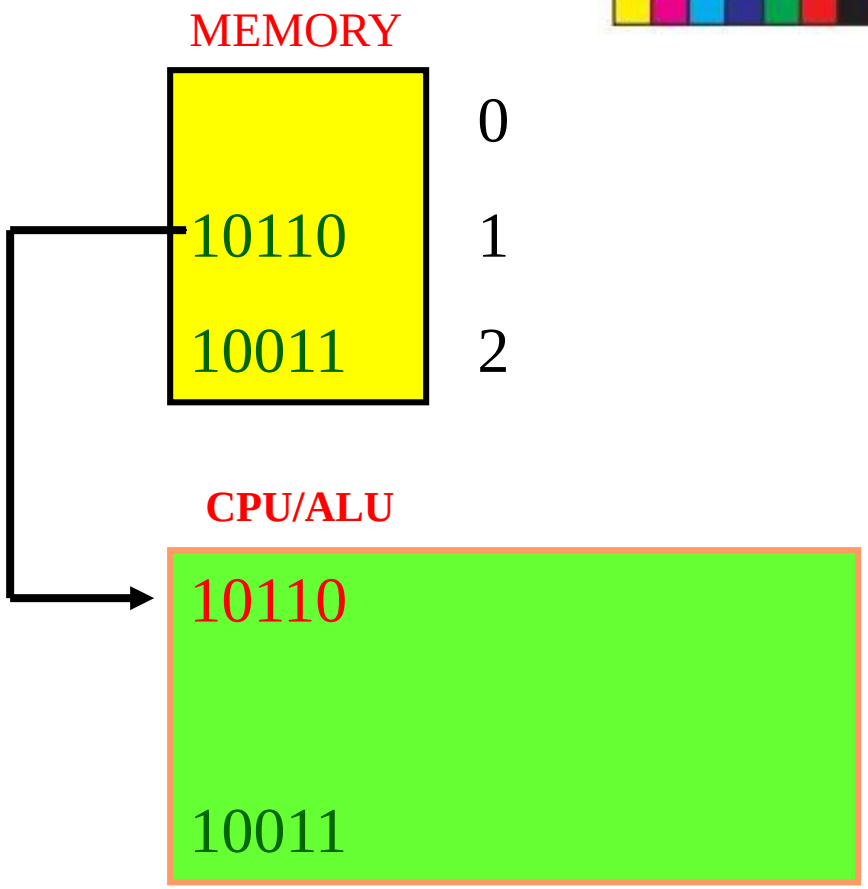
+10011

———————

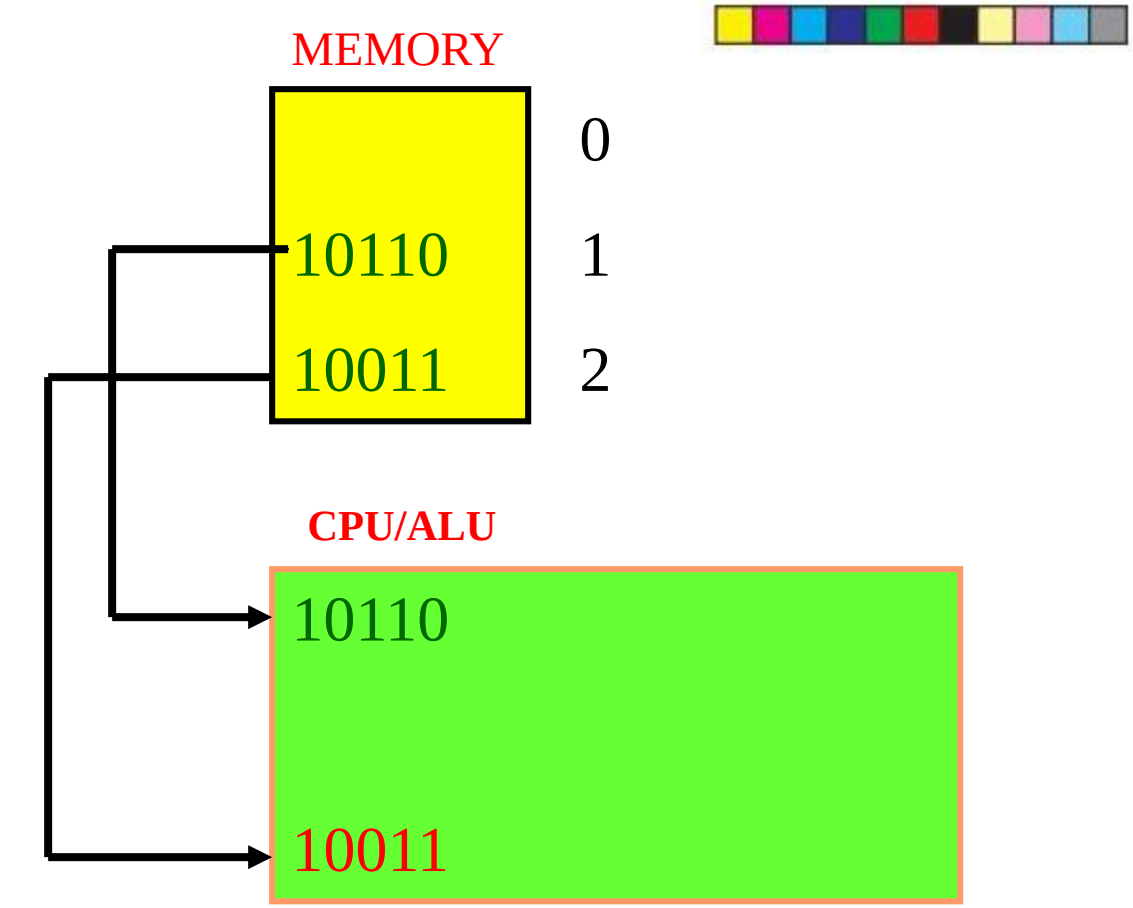
101001 ← Result

MSBit LSBit

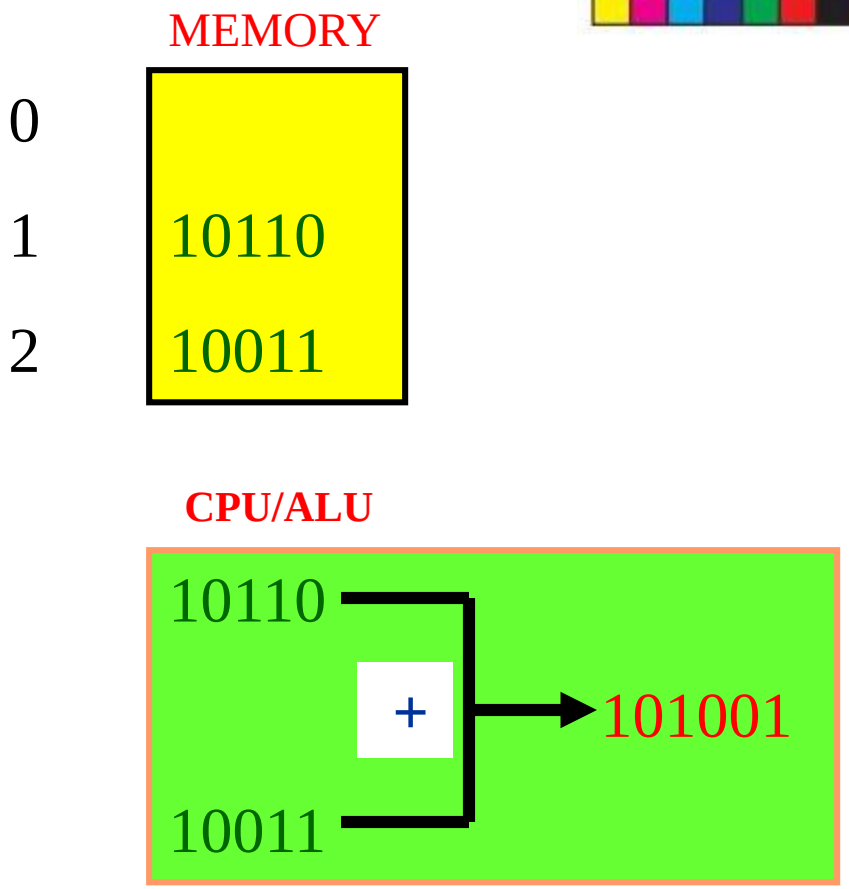
Add two binary numbers



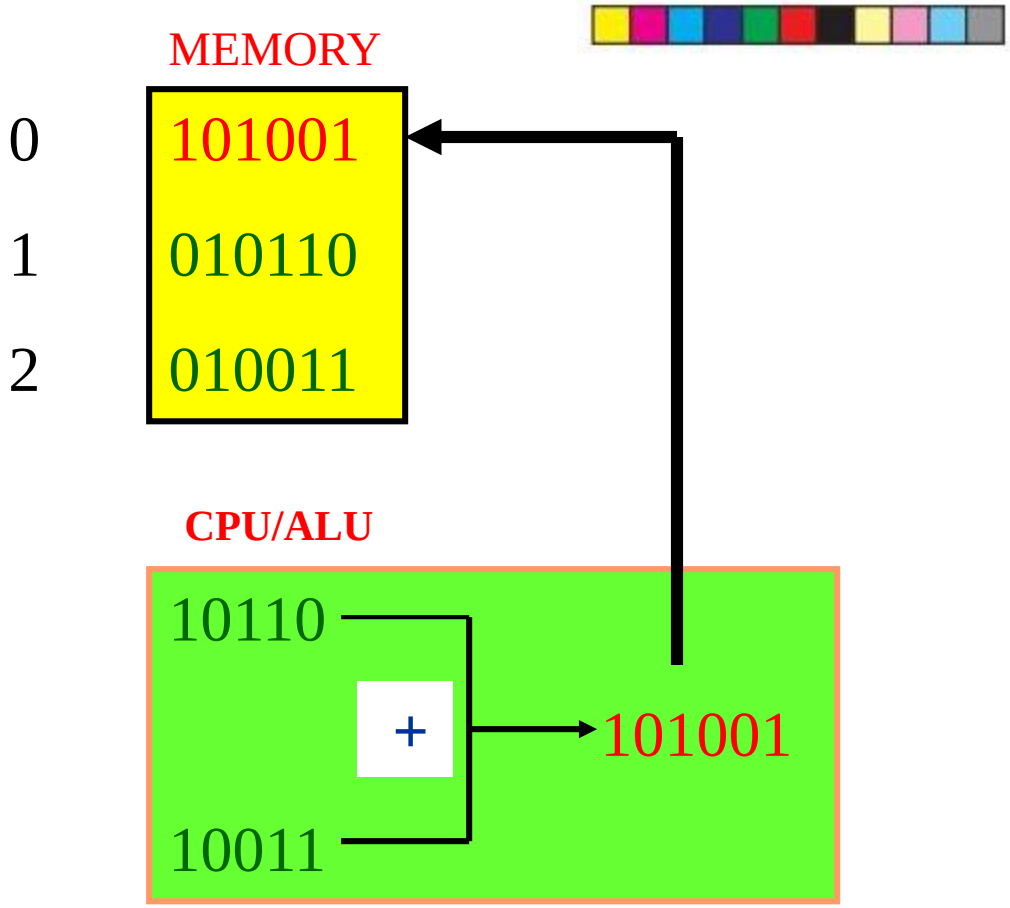
Add two binary numbers



Add two binary numbers



Add two binary numbers



10011, 10110 and 101001 are stored in the CPU/ALU in Registers

Addition using Octal (1)

- 1476
- +3554

  ????

Addition using Octal (2)

- 1476 6+4 = 10 => (1, 2)
- +3554 carry sum

1

1476

3554

2

Addition using Octal (3)

- 1476 1+4+5 = 10 => (1, 2)
- +3554 carry sum

111

1476

3554

252

Addition using Octal (4)

- 1476 1+1+3 = 5 => (0, 5)
- +3554 carry sum

111

1476

3554

5252

Addition using Hex (1)

- 59F
- +E46

???

Addition using Hex (2)

- 59F F + 6 = 15 + 6 => (1, 5)
- +E46 carry sum

1

59F

E46

5

Addition using Hex (3)

- 59F 1 + 9 + 4 = 14 => (0, E)
- +E46 carry sum

```
  01
  59F
  E46
 ----
  E5
```

Addition using Hex (4)

- 59F 0 + 5 + E = 19 => (1, 3)
- +E46 carry sum

01

59F

E46

13E5

Subtraction

Decimal subtraction

$$\begin{array}{r} 401 \\ -\ 32 \\ \hline 369 \end{array}$$

Binary subtraction

```
  0
- 0
---
  0
```

Binary subtraction

0	1
- 0	- 0
0	1

Binary subtraction

0	1	1
- 0	- 0	- 1
0	1	0

Binary subtraction

0	1	1	0
- 0	- 0	- 1	- 1
0	1	0	1 = result
			1 = borrow

1 0 1 1 0 = 2+4+16 = 22

\- 1 0 0 1 1 = 1+2+16 = 19

---

(1)

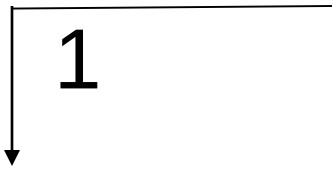
Since 0-1

1 0 1 1 0

- 1 0 0 1 1

(a)

Borrow 1 from next (left) column



1

1 0 1 1 0

- 1 0 0 1 1

(b)

Borrow 1 from next (left) column

```
      1
    1 0 1 1 0
  - 1 0 0 1 1
  -----------
          1
```

Therefore 10-1 = 1

(c)

Since we borrowed a 1 a 0 is left

$$\begin{array}{rcccc} & & & 0 & 1 \\ & & & \nearrow & \\ & 1 & 0 & \cancel{1} & 0 \\ - & 1 & 0 & 0 & 1 \\ \hline & & & & 1 \end{array}$$

(d)

Since 0 – 1, borrow 1 from next (left) column

			0	1
	1	0	1	~~1~~ 0
-	1	0	0	1 1
				1

(e)

Borrow 1 from next (left) column

```
          1 0 1
          ↗
  1   0   1   1   0
- 1   0   0   1   1
---------------------
              1
```

(f)

Borrow 1 from next (left) column

1 0 1

1 0 1 ~~1~~ 0

- 1 0 0 1 1

---

1 1

Therefore 10-1 = 1

(g)

Since we borrowed a 1 a 0 is left

```
          0
    1  0  1  1  0
  - 1  0  0  1  1
  ---------------
          1  1
          ↑
```

(h)

Since we borrowed a 1 a 0 is left

0

1 0 1 1 0

\- 1 0 0 1 1

\-----------------------

0 1 1

Therefore 0-0 = 0

(i)

1 0 1 1 0

- 1 0 0 1 1

---

0 0 1 1

0-0 = 0

(j)

1 0 1 1 0

- 1 0 0 1 1

---

0 0 0 1 1

↑

1-1 = 0

Result: 22-19=3

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0 \\ -\ 1\ 0\ 0\ 1\ 1 \\ \hline 0\ 0\ 0\ 1\ 1 \end{array}$$

Multiplication and Division

Multiplication

- Shift + Add

Binary Multiplication

```
    101
  X 100
  -----
    000        100 add zeros
   000         100 shift-add zeros
+ 101          100 shift-copy number
  -----
  10100        Add
```

Binary Multiplier

- We need to design
  - An Adder
  - A Shifter

Next course CMPT281 …

Subtraction…2's complement

- Signed numbers are used to simplify the binary subtraction operation …..
- … and to remove the need to have an extra character (the minus) to represent negative numbers (signed representation).
- After all in computing the **Binary System** is used and not a **Trinary System** (1-0-minus) …

Sub …1000100-1010100, using 2's comp.

- M = 100 0100 = 68
- -N = -101 0100 = -84

-16

Step-1: Find the 2's complement of the negative number (-84)

Sub …1000100-1010100, using 2's comp.

- M = 100 0100 = 68
- -N = -101 0100 = -84

-16

2's of 1010100 ?

1's = 0101011

+ 1

010 1100

Step-2: ADD the 2's complement of the negative number (-84) with the positive number (68)

Sub ...1000100-1010100, using 2's comp.

- M = 100 0100 = 68
- -N = -101 0100 = -84

-16

Therefore,

2's of 1010100 ?

1's = 0101011

+ 1

010 1100

100 0100

+ 010 1100

NC 111 0000

Step-3: If there is no Carry-out the result is NEGATIVE and in 2's Complement form

Sub …1000100-1010100, using 2's comp.

- M = 100 0100 = 68
- -N = -101 0100 = -84

-16

Therefore,

2's of 1010100 ?

1's = 0101011

+ 1

010 1100

100 0100

+ 010 1100

NC 111 0000

Step-4: Since we use unsigned arithmetic …place a negative sign in front of the number

The answer is: - 2's of (111 0000) = -10000 = $(-16)_{10}$