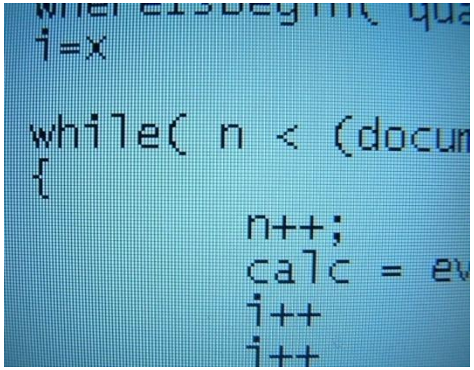
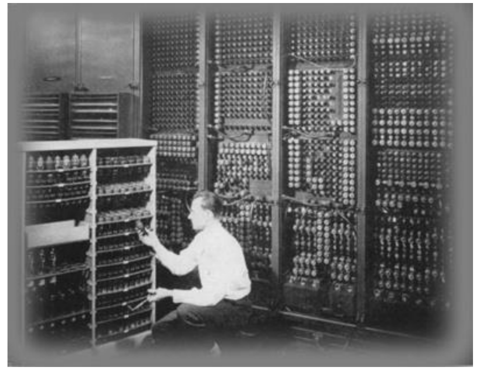




CS1PR16

Introduction to Computer Architecture

Learning Objectives

- Describing the processing of a generic program on a computer system and the interplay between ALU, CPU and Memory
- Executing a sequence of instructions step by step via an instruction pointer
- Converting decimal numbers into binary numbers
- Encoding/decoding binary numbers as ASCII character set

Outline

- Data Representation
  - Bits and Bytes
  - Binary addition
  - Negative numbers
  - Operations: binary subtraction
  - Strings
  - Floating-point numbers
- Computer Architecture
  - Components
  - Execution of “code”

Bits and Bytes

- Computers operate on binary digits (bits)
  - Bits, being of value 0 or 1
  - Consider an electrical switch being on or off
- A byte is generally a term for a collection of 8 bits
  - Gives rise to 2^8 possible combinations (Byte is also called octet)
- How do we store whole numbers (integers)?
  - i.e., an 8-bit number has 256 possible values
  - Byte = 8 Bits = $[b7\ b6\ b5\ b4\ b3\ b2\ b1\ b0]_2$
    - Uses the base-2 numeral system! (indicated by **subscript 2**)
    - For the decimal system, we typically omit the "base" (10)
  - $[00000000]_2 = [0]_{10}$
  - $[00000001]_2 = [1]_{10}$
  - $[11111111]_2=[255]_{10}$ = Maximum = $2^8 - 1$
- Use more bits (e.g. 16, 32, 64) to create larger numbers

Integers

- To calculate the decimal equivalent of a binary number, just like decimal, consider each column in increasing magnitude:
  - .. 2^6 2^5 2^4 2^3 2^2 2^1 2^0
  - .. 64 32 16 8 4 2 1
- Then sum up each column that has a 1
- For example, $[01100100]_2$
  - 0 1 1 0 0 1 0 0_2
  - 128 64 32 16 8 4 2 1 = 64+32+4=$[100]_{10}$

Integers

University of Reading

- Value(byte) = $\sum_{i=1}^{8} b_i \, 2^i$
  - Base 2
  - Similar to the decimal system that just uses as base 10^i
- Converting between binary and decimal system:
  - Example: $[0101\ 1001]_2$ =1*1+0*2+0*4+1*8+1*16+0*32+1*64+0*128

    =1+8+16+64

    = $[89]_{10}$
- Other representations (bases) are frequently used:
  - Hexadecimal (base 16)
  - Octal (base 8)
  - You will learn more about this in another lecture!

Binary Addition

- The rules for binary addition
- Add each weighted column up and include any carry (bit):
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 0 = 1$
  - $1 + 1 = [10]_2$, i.e. 0 with a carry into the column to the left
  - $1 + 1 + 1 = [11]_2$, i.e. 1 with a carry into the column to the left

Binary Addition

- Example: add 45_{10} to 122_{10} using 8 bit binary
  - $45_{10} = 00101101_2$
  - $122_{10} = 01111010_2$

$$
\begin{array}{lr}
 & 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1_2 \\
 & +\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0_2 \\
 & =\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1_2 \\
\hline
\text{Carry} & 1\ 1\ 1\ 1\
\end{array}
\quad = 128+32+4+2+1=167_{10}
$$

Negative numbers

- By default, numbers are generally treated as signed
- Signed and unsigned are often keywords in programming
  - So far we dealt with unsigned, e.g., 00000000_2.. 11111111_2 = $0..255_{10}$
- A signed number needs to store somehow that it can be negative
  - Could be done using a bit to indicate if it is + or – (e.g., 1 = -)
    - Example: $[-1 + 1]_{10}$ = $[1000\ 0001 + 0000\ 0001]_2$ should be zero!
    - => problem: addition of number requires check
  - The 2's complement representation avoids this check
- Signed representation using 2's complement number
  - 10000000_2.. 01111111_2 = -128 … 0 … +127

2's Complement

- We use 2's complement to describe signed numbers
  - In 2's complement, to calculate the binary for a negative number:
    - Start with its positive binary equivalent
    - Invert all bits (Technically, compute "1 – bit")
    - Add 1
  - Benefit: you can add positive or negative bits the same way!
- Example, calculate -100_{10} in binary to 8 bits
  - $+100_{10}$ = 01100100_2
  - 01100100 inverted = 10011011
  - 10011011 add 1 = 10011100_2
- The context of the program decides if it is (un)signed

  Our example number can mean in decimal: -100_{10} or 156_{10}

Subtracting Binary

- Computers do not have subtraction circuits!
- To subtract, convert a number to its negative and then add
  - i.e. 45-22 = 45 + (-22) = 23

+45 is 00101101_2

-22 is 11101010_2

```
          0  0  1  0  1  1  0  1
        + 1  1  1  0  1  0  1  0
carry=1 1  1  0  1
        = 0  0  0  1  0  1  1  1  =  23₁₀
```

Here, the extra bit is silently disposed.

Two positive high numbers would actually lead to a meaningful overflow = carry bit.

Group Work

Task

Perform the following operation in binary:

18 – 4 = ?

How does the 2’s complement of 0 look like?

Time: 3 min

Share: 1 min

In 2's complement, to calculate the binary for a negative number:

- *Start with its positive binary equivalent*
- *Invert all bits (Technically, 1 – each bit)*
- *Add 1*

Strings

- A character like 'a' is a number
- A text is just a sequence of numbers
  - Individual numbers can be stored to create a string:

    'h' 'e' 'l' 'l' 'o' -> "hello"
- Somehow we must map the number to the character
  - This is done by an encoding
- How does the computer know when the string ends?
  - Either store the length somewhere or have a special "terminator"
  - All strings end with the ASCII character '\0' or $[0]_{10}$

The ASCII Character Set

- **The American Standard Code for Information Interchange (ASCII)** is a character encoding
- Defines how to map binary data to readable characters (glyphs)
  - Interpretation of the digital data
  - Human-readable and machine-readable
- ***A glyph*** *is an elemental symbol within an agreed set of symbols, intended to represent a readable character for the purposes of writing* [Wikipedia]
- The original ASCII-table uses 7 bits
  - The 8^{th}-bit is language-specific
  - Adds most important glyphs for a language, e.g., üöä for German

ASCII Table: Mapping

University of Reading

The original table defines how to map the 7 bits to a character:

USASCII code chart

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII Table: Mapping

The original table defines how to map the 7 bits to a character:

- Examples:
  - 'A' = $[0100\ 0001]_2 = 64 + 1 = 65$
  - 'Z' = $[0101\ 1010]_2 = 64 + 16 + 8 + 2 = 92$
- Control characters are special
  - Example: newline (carriage return) often written as '\n' is 13
- For best compatibility, use only these 128 characters!

The table rows cover the four lower bits

The columns the high three bits

=> A

=> Z

USASCII code chart

Bits				$b_7 b_6 b_5$ →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

ASCII Table: Another Representation

University of Reading

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

From www.asciitable.com

Group Work

University of Reading

Task

Using the ASCII table, write down your first name as

1) the sequence of decimal numbers
2) the sequence of binary representation

Time: 3 min

Share: 2 min

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

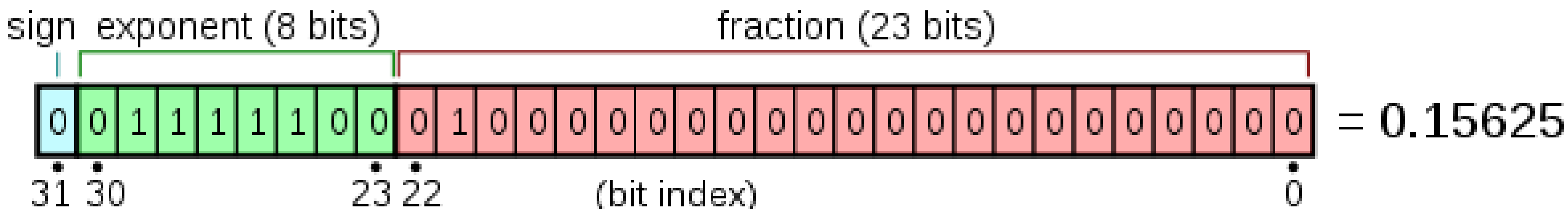
Source: www.LookupTables.com

Floating-Point Numbers

- How can we express a real number like 3.45?
  - Or 100 trillions?
  - 32-bit: 2^{32}: 0..4,294,967,295 – 4 billion may still not be enough
- The IEEE standard 754 defines a single-precision 32-bit (float) and 64-bit (double) precision binary number format
  - Includes –∞ and +∞, but also -0 and NotANumber(NAN)
  - This type of format is called floating-point because "." can be moved
- The ALU of our CPU has arithmetic units that understand
  - 2's Complement but also these IEEE standard formats

Floating-Point Numbers

- Single precision floating-point numbers contain
  - a single bit sign bit
  - an 8-bit exponent
  - a 23-bit mantissa (fraction part)



$$\text{value} = (-1)^{\text{sign}} \times 2^{(e-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i}\, 2^{-i}\right)$$

- It is not trivial to create a circuit that does proper math
- Floats are representing scientific numbers, e.g. 3.428e9
  - Can represent numbers in the region of ±1.4e−45 to ±3.4e38

Floating-Point Numbers

- double precision contain
  - a single bit sign bit
  - an 11-bit exponent
  - a 52-bit mantissa
  - Can represent numbers in the region of

    $\pm 2.2250738585072020e{-308}$ to $\pm 1.7976931348623157e308$
  - Does require twice the memory as float
- Note that floating-point numbers are an approximation
  - Stores only a number of significant digits: rounded value
  - They are not associative: (a+b)+c != a+(b+c)
    - As they are rounded

File System

- A computer manages data in file systems
  - Organizes data into directories that contain files/directories
  - Hierarchical organization
- Example hierarchy:
  - *Dir*
    - *Subdir1*
      - *file1.txt*
      - *anotherfile.txt*
    - *Subdir2*
      - *SomethingElse.txt*
- A file is just a sequence of binary data (with some permissions)
  - When written, it has a given size

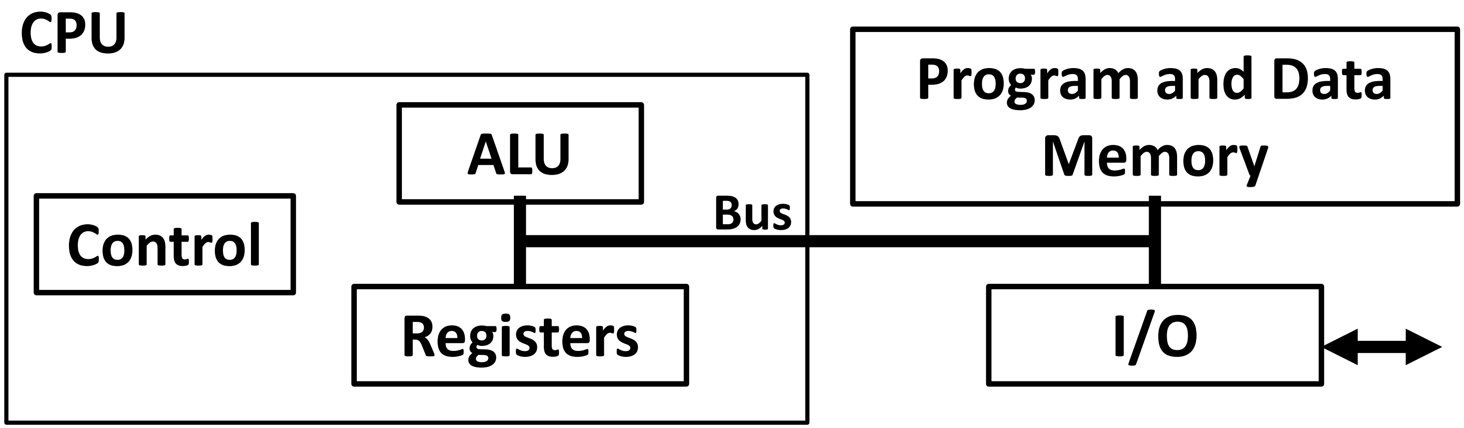
Computer Architecture

University of Reading

- The Central Processing Unit (CPU) forms the "brains"
  - An integrated programmable microprocessor
    - 8, 16, 32, 64, even 128 bit
  - Provides limited storage (so-called registers)
  - Provides Arithmetic and Logic Unit (ALU)
    - Manipulates registers
- The instruction set defines operations
  - Instructions perform operations
  - Tell the processor what to do (programmable)
  - The CPU manufacturer provides a manual
- A program specifies the instructions to run
  - A program is created by humans (most likely)
  - A program is executed by a computer that understands binary
- Assembler: language with a direct translation from human-readable instructions to a binary representation in CPU the instruction set

First Microprocessor by Intel [Wikipedia]

Von Neumann Architecture



- Instructions are just "data", memory stores more data
  - Data is transferred via a joint bus between CPU and memory
- Control decodes instructions and determines what to do
  - Based on the instructions, controls memory access as well
- The ALU performs the arithmetical and logical operations
  - Stores the results in the registers

The ALU

- The ALU performs the arithmetical and logical operations
  - Typically using the registers as input and for storing the result
- Arithmetic:
  - Operations: +, -, *, /, %,
  - For floating point numbers and for signed/unsigned numbers
- Logical operations:
  - Compare two numbers: x <= y
  - Bitwise operations, i.e., manipulate bytes or individual bits
- Other operations may be provided

ALU: Logical Bitwise Operators

- Bitwise operators apply logical operators on each bit
  - Allows to manipulate bits of bytes
- Take two Bytes a = $[a7 \dots a0]_2$ and b = $[b7 \dots b0]_2$
  - Result c = $[c7 \dots c0]_2$ where
  - Combine c7 = a7 <op> b7, a6 <op> b6, ..., a0 <op> b0
- And: a & b
  - Result: ci = 1 if ai = 1 AND bi = 1 (0 otherwise)
- Or: a | b
  - Result: ci = 1 if ai = 1 OR bi = 1 (or both)
- Xor: a ^ b
  - Result: ci = 1 if one of ai or bi is 1
- Complement: ~a (inverts data)
  - Result: ci = 0 if ai = 1 and ci = 1 if ai = 0

Bitwise Operators

AND

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1

Results of combining two bits with the bitwise AND operator &

OR

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	1

Results of combining two bits with the bitwise OR operator |

XOR

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	0

Results of combining two bits with the bitwise exclusive OR operator ^

Computer Architecture

- Main memory
  - As the microprocessors internal memory is small, we have external memory (RAM)
  - Can be thought of as an array of digital information with locations

    data = [128, 256, 128, …]

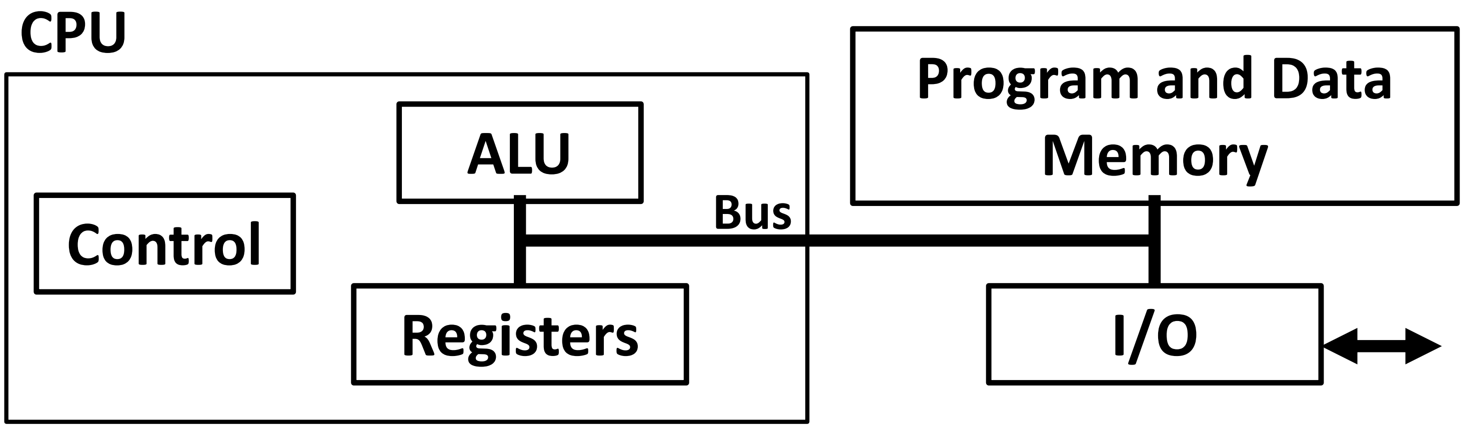
    location = 1 2 3 ….
- Memory to store code and data
  - Random Access Memory (RAM)
  - Cache inside the processor (fast!)
  - Read-Only Memory (ROM)
  - Disc storage (magnetic and optical)
- Input/Output
  - Networking
  - Buses
  - Human-Computer Interfaces

Computer Architecture - Registers

- A register is a small local data store inside the CPU
- Data Registers
  - Working space to perform operations on
  - For storing data to/from memory
  - For storing data from ALU
- Status Registers - information about last computation
  - 'flags' for carry over of math, overflow, zero, negative, …
- Address Registers
  - Memory address of the current program location
  - Stack pointer - has the address of the top of the stack memory (later)
  - Instruction pointer …

Architectures

- The Von Neumann Architecture
  - has a shared **bus** for data and instructions
    - data and instructions are treated similarly



- The Harvard Architecture has a separate bus for instructions

Assembler

Example for adding two numbers:

a = 3

b = 4

c = a + b

This operation can be told to the microprocessor using assembler:

```
movl $3, 12
movl $4, 8
movl 12, %edx
movl  8, %eax
addl %edx, %eax
movl %eax, 4
```

Explanation

Store number 3 into memory pos. 12

Store number 4 into memory 8

Move memory 12 to register EDX

Move memory 8 to register EAX

Add EDX and EAX and store into EAX

Store register EAX into memory 4

Assembler

The instructions are processed sequentially

The CPU has an instruction pointer (IP) which specifies what instruction to run next

IP →

```
movl $3, 12
movl $4, 8
movl 12, %edx
movl  8, %eax
addl %edx, %eax
movl %eax, 4
```

Store number 3 into memory 12

Store number 4 into memory 8

Move memory 12 to register EDX

Move memory 8 to register EAX

Add EDX and EAX store into EAX

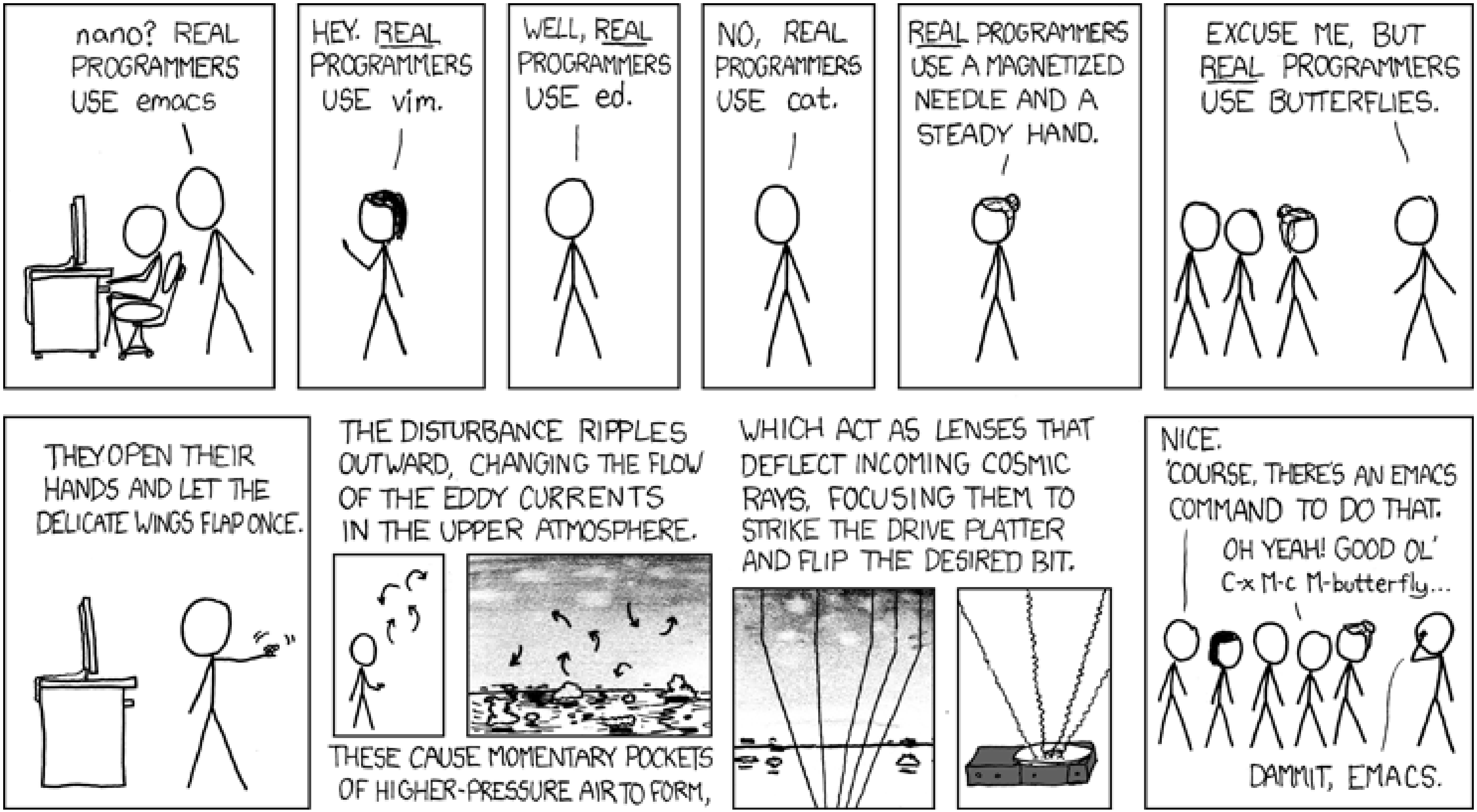
Store register EAX into memory 4

A multicore processor has one IP per core

- Actually, the IP contains the memory location for the next instruction to run
- Instructions are nothing more than binary data, stored in memory!

Fun: Since you now worked with Linux

University of Reading



Source: XKCD.COM

Summary

- A computer stores data in digital representation
  - Bits and Bytes
  - Whole numbers
  - Floating-Point Numbers
  - Encoding + ASCII determines the interpretation of glyphs
- A CPU processes instructions given by a program
  - The ALU performs the arithmetic and logical operations
  - Operations: addition, inversion
  - Supports: 2's complement, floating-point
  - Logic operations: AND, OR, NOT, XOR …

SESTEM PROJECT

- Aims to understand the experiences of undergraduate students in STEM degrees, especially those who self-consider as from ‘Black, Asian and Minority Ethnic’ (BAME) backgrounds.
- The goal is to develop strategies or resources to reduce the differential degree outcome (sometimes referred to as the ‘attainment gap’) and improve the experiences of students from different backgrounds at university.
- Receive up to £60 in Amazon e-vouchers for writing 2 termly reflections, attending an interview and taking part in a workshop in the summer term (lunch provided!).

SESTEM@READING.AC.UK

Reham Elmorally
r.elmorally@pgr.reading.ac.uk

Meggie Copsey-Blake
megan.copsey-blake@student.reading.ac.uk