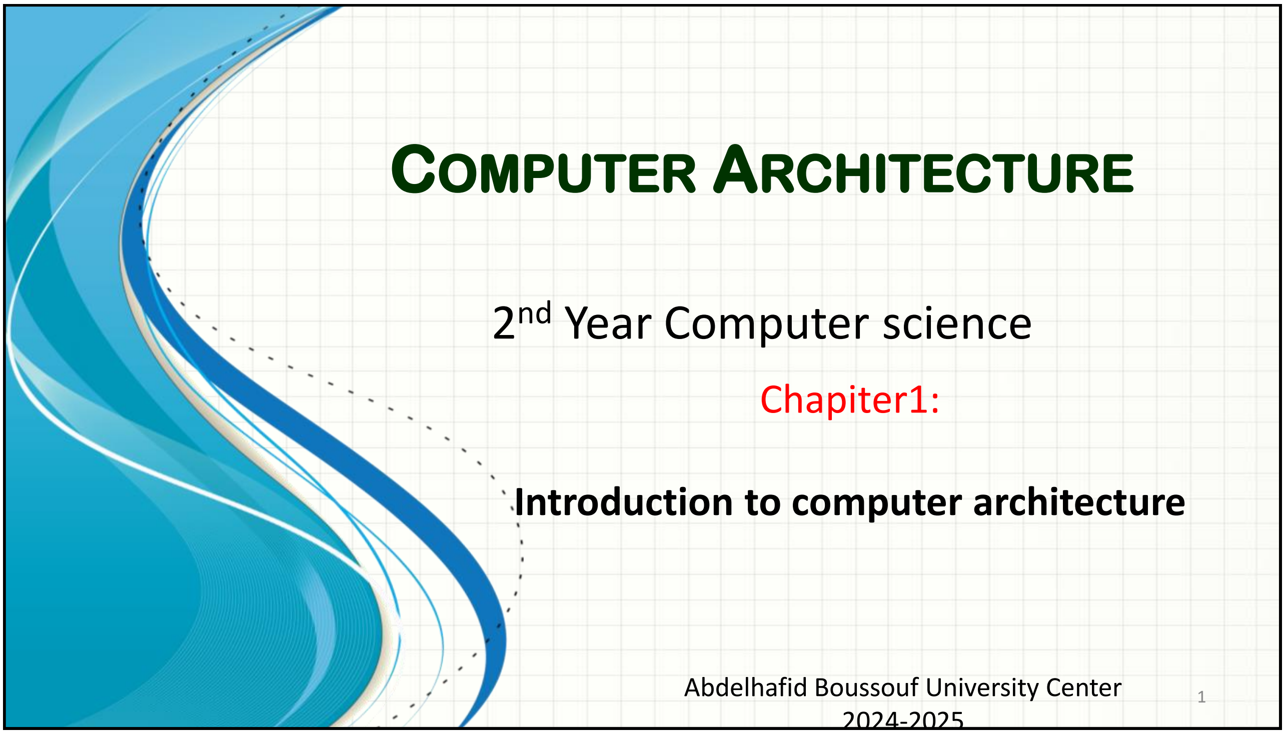


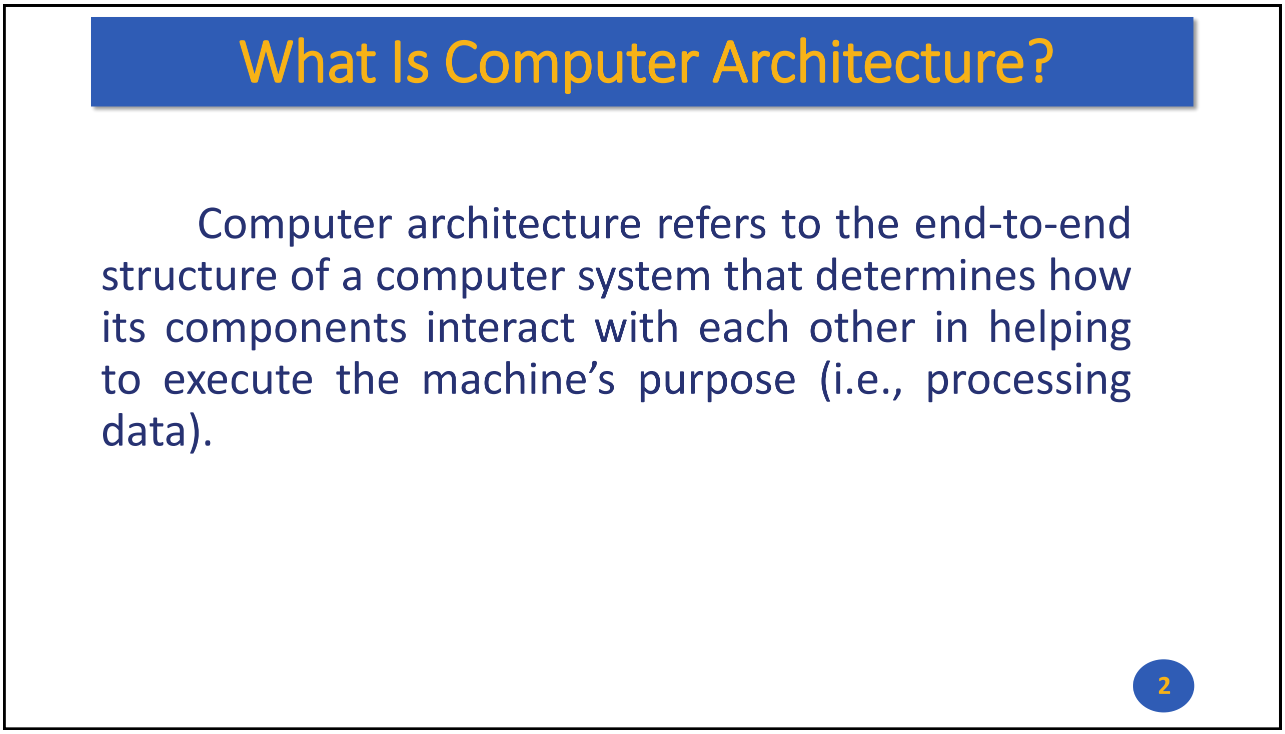
COMPUTER ARCHITECTURE

2nd Year Computer science

Chapiter1:

Introduction to computer architecture

Abdelhafid Boussouf University Center
2024-2025



What Is Computer Architecture?

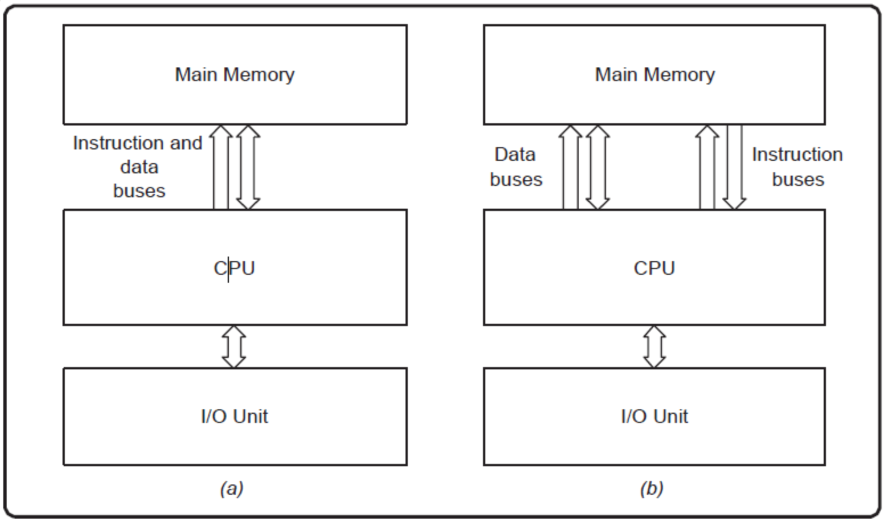
Computer architecture refers to the end-to-end structure of a computer system that determines how its components interact with each other in helping to execute the machine’s purpose (i.e., processing data).

What Is Computer Architecture?

The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.

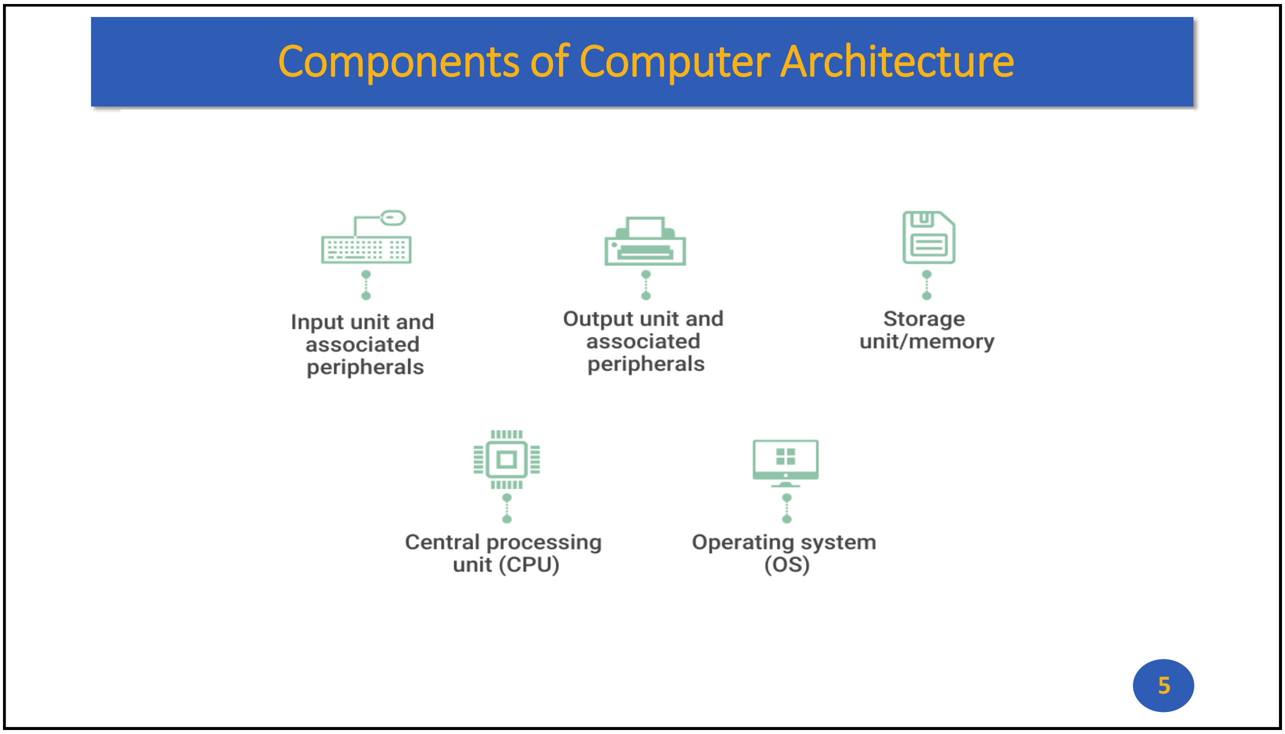
3

Von Neumann Architecture and Harvard Architecture



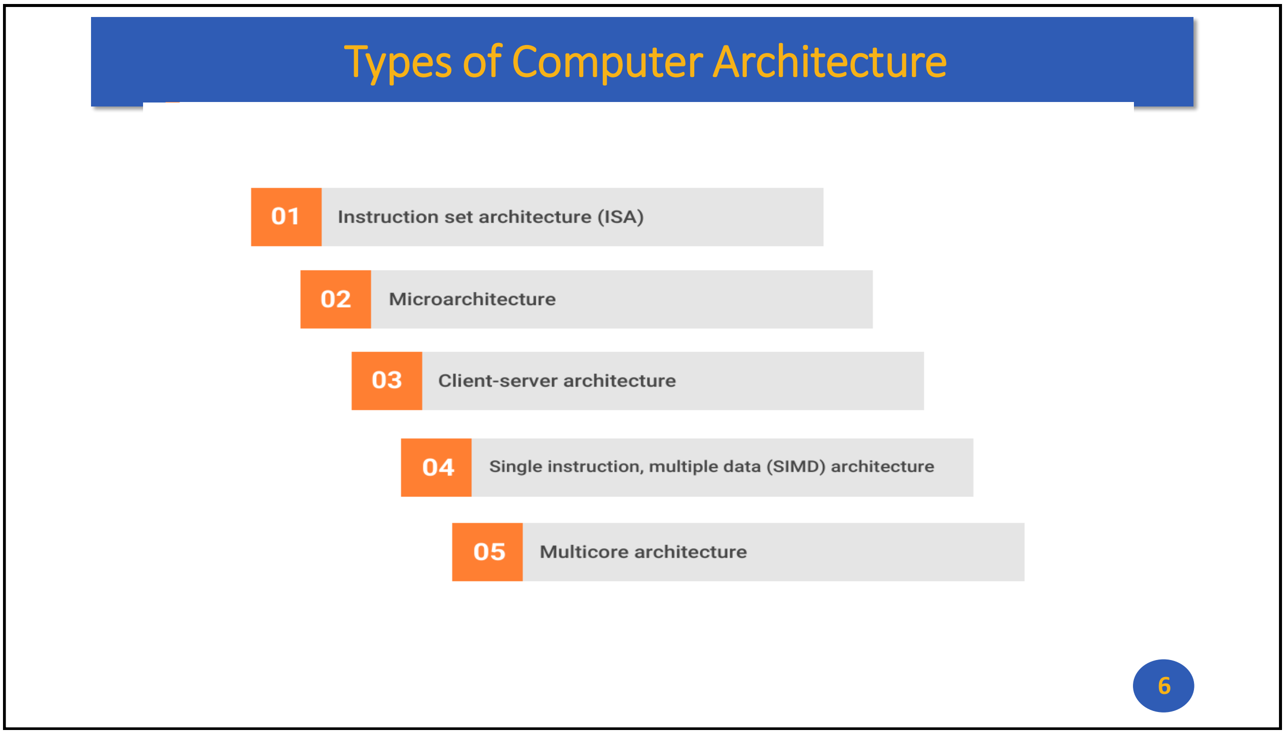
Examples of Computer Architecture: Von Neumann Architecture (a) and Harvard Architecture (b)

4



Components of Computer Architecture

- Input unit and associated peripherals
- Output unit and associated peripherals
- Storage unit/memory
- Central processing unit (CPU)
- Operating system (OS)



Types of Computer Architecture

01 Instruction set architecture (ISA)

02 Microarchitecture

03 Client-server architecture

04 Single instruction, multiple data (SIMD) architecture

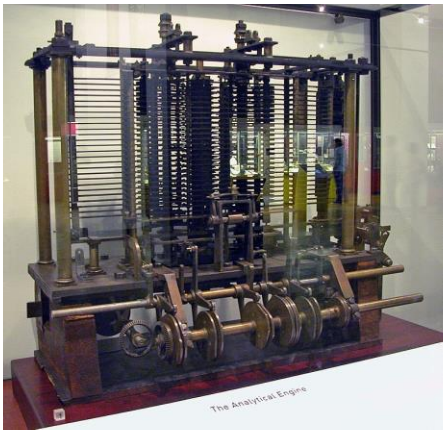
05 Multicore architecture

What affects performance?

Hardware/Software Component	How It Affects Performance
Algorithm	Determines both the number of source-level statements and the number of I/O operations executed
Programming Language, Compiler, and Architecture	Determines the number of computer instructions for each source-level statement
Processor and Memory System	Determines how fast instructions can be executed
I/O System (Hardware and Operating System)	Determines how fast I/O operations may be executed

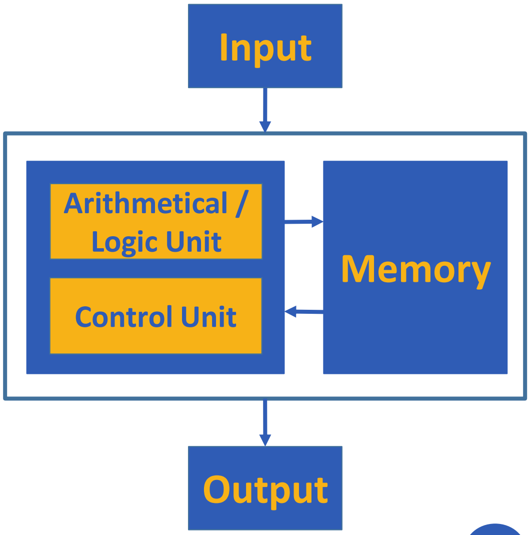
History: 0th Generation – Mechanical

- 1834–71: Analytical Engine designed by Charles Babbage
- Mechanical gears, where each gear represented a discrete value (0-9)
- Programs provided as punched cards
- Never finished due to technological restrictions



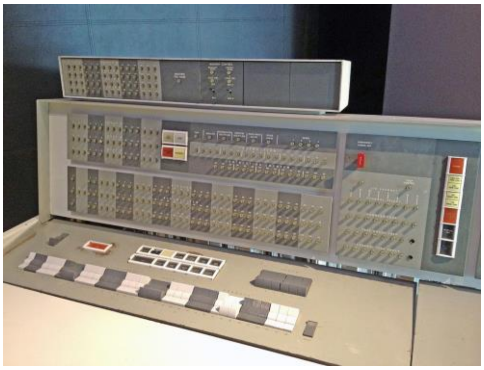
History: 1st Generation - Vacuum Tubes

- 1945–55: first machines were created (Atanasoff–Berry, Z3, Colossus, ENIAC)
- All programming in pure machine language
- Connecting boards and wires, punched cards (later)
- Stored program concept



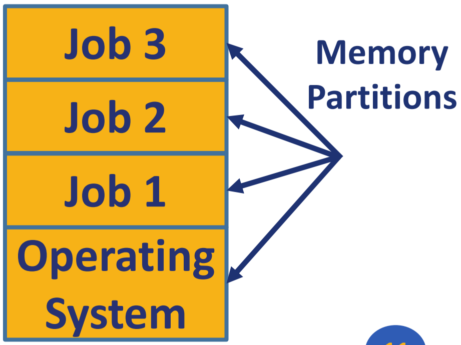
History: 2nd Generation - Transistors

- 1955–65: era of mainframes (e.g. IBM 7094) used in large companies
- Programming in assembly language and FORTRAN
- Batch systems (IO was separated from calculations)
- Punched cards and magnetic tape
- Loaders (OS ancestors)



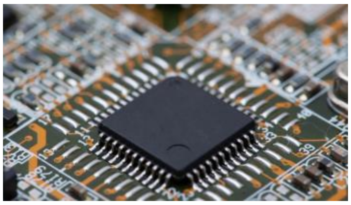
History: 3rd Generation – Integrated Circuits

- 1965–1980: computer lines using the same instruction set architecture (e.g. IBM 360)
- First operating systems (e.g. OS/360, MULTICS)
- Multiprogramming and timesharing
- Computer as utility
- Programming languages and compilers (LISP, BASIC, C)



History: 4th Generation – VLSI and PC

- 1980–Present: personal computers, laptops, servers (Apple, IBM, etc.)
- Architectures: x86-64, Itanium, ARM, MIPS, PowerPC, SPARC, RISC-V, etc.
- Operating systems: UNIX (System V and BSD), MINIX, Linux, MacOS, DOS, Windows (NT)
- ISA (CISC, RISC, VLIW), caches, pipelines, SIMD, vectors, hyperthreading, multicore

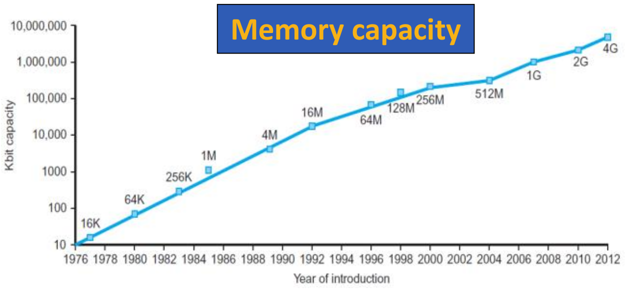


History: 5th Generation – Mobile devices

- 1990–Present: mobile devices, embedded systems, IoT devices
- Custom processors and FPGAs
- Mobile operating systems: Symbian, iOS, Android, Windows Mobile
- Real-time operating systems

Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

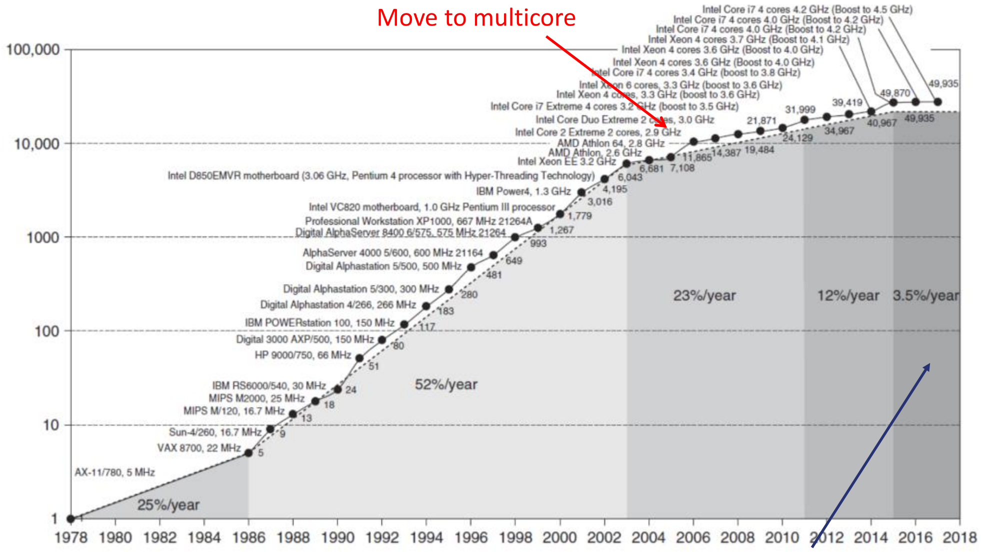


Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Moore’s Law

- Gordon Moore (1929-...) cofounded Intel in 1968 with Robert Noyce
- **Moore’s Law:** number of transistors on a computer chip doubles every year (observed in 1965)
- Limited by power consumption
- Slowed down since 2010

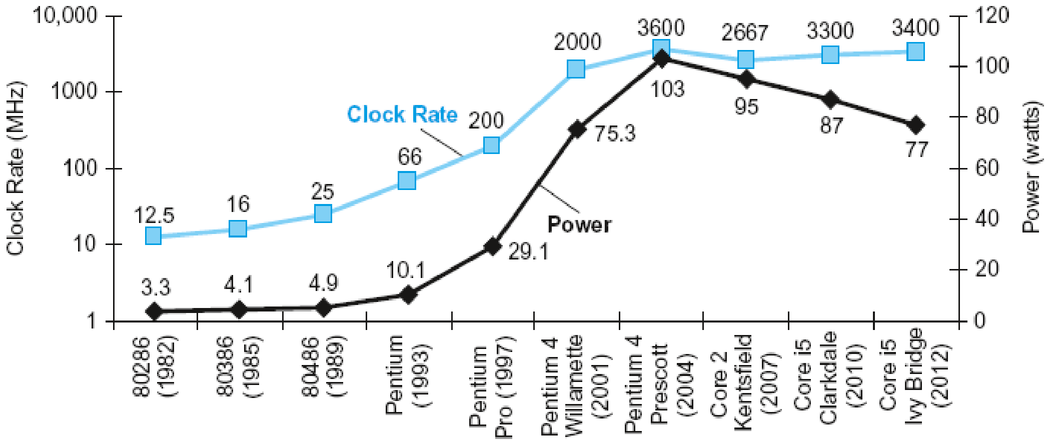
Single Core Performance



Move to multicore

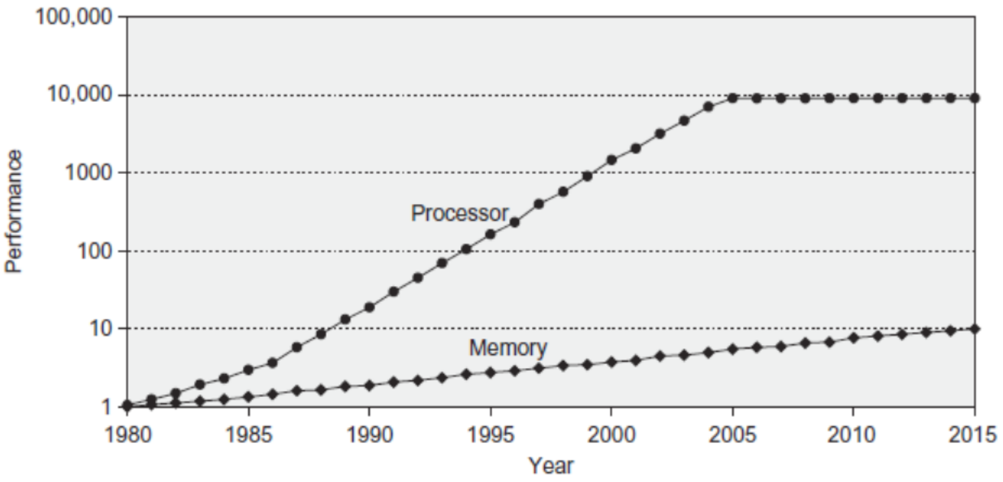
Constrained by power, instruction-level parallelism, memory latency

Power Trends



17

Memory Performance Gap



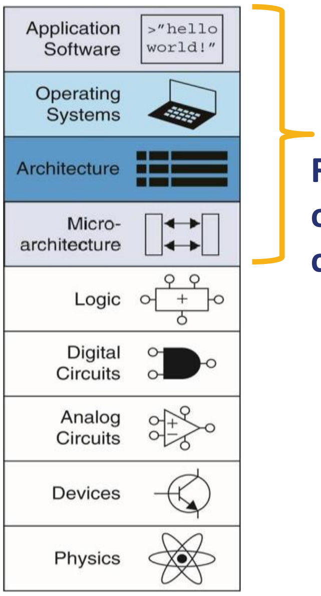
18

Current Challenges

- Single core performance improvement has ended
  - More powerful microprocessor might not help
- Memory-efficient programming
  - Temporal locality
  - Spatial locality
- Parallelism to improve performance
  - Data-level parallelism
  - Thread-level parallelism
  - Request-level parallelism
- Performance tuning require changes in the application

Concluding Remarks

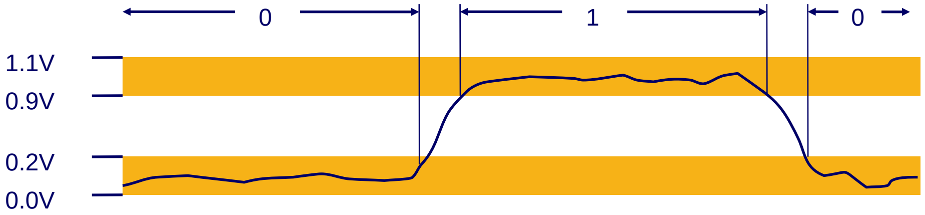
- To create software that efficiently deals with big data, we need to understand how hardware is organized and managed by operating system
  - Computer architecture
  - Assembly language
  - Compiler basics
  - Operating systems



Focus of this course

Everything is Bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- Why bits? Electronic implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



Number Systems

- Decimal numbers

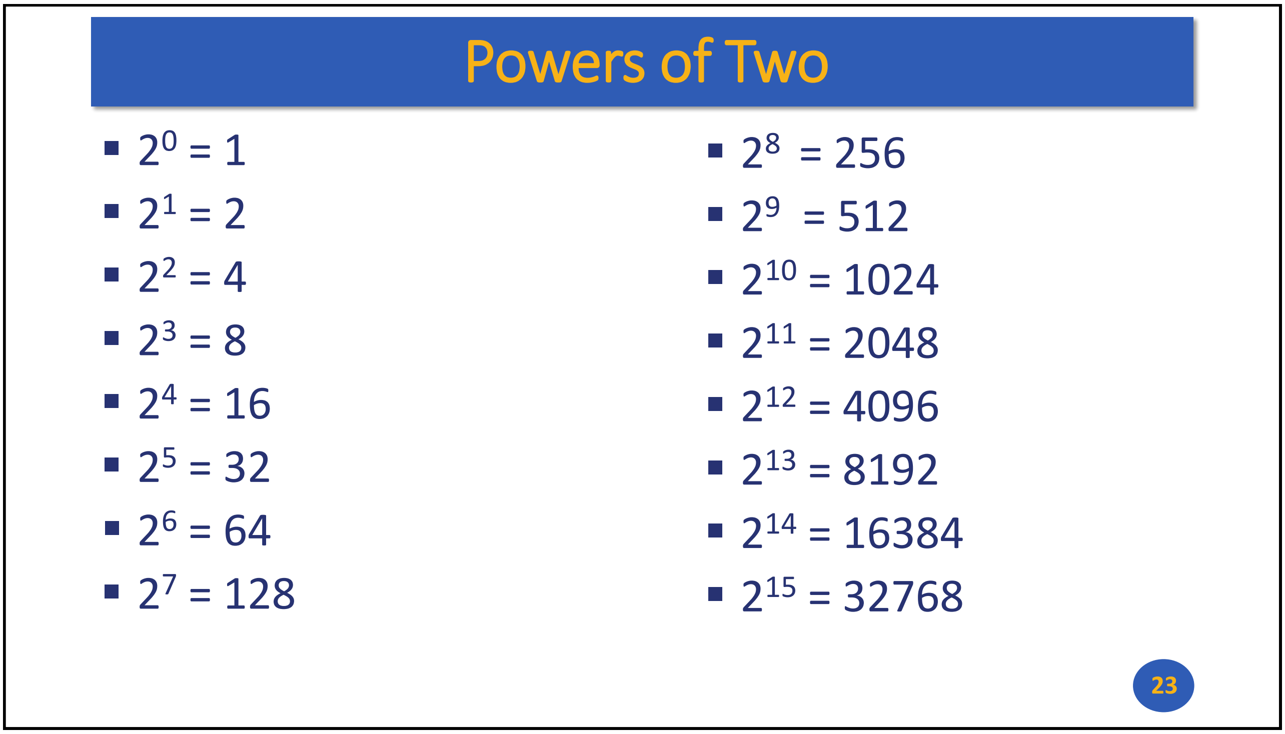
1000's column
100's column
10's column
1's column

$$5374_{10} = \underbrace{5 \times 10^3}_{\text{five thousands}} + \underbrace{3 \times 10^2}_{\text{three hundreds}} + \underbrace{7 \times 10^1}_{\text{seven tens}} + \underbrace{4 \times 10^0}_{\text{four ones}}$$

- Binary numbers

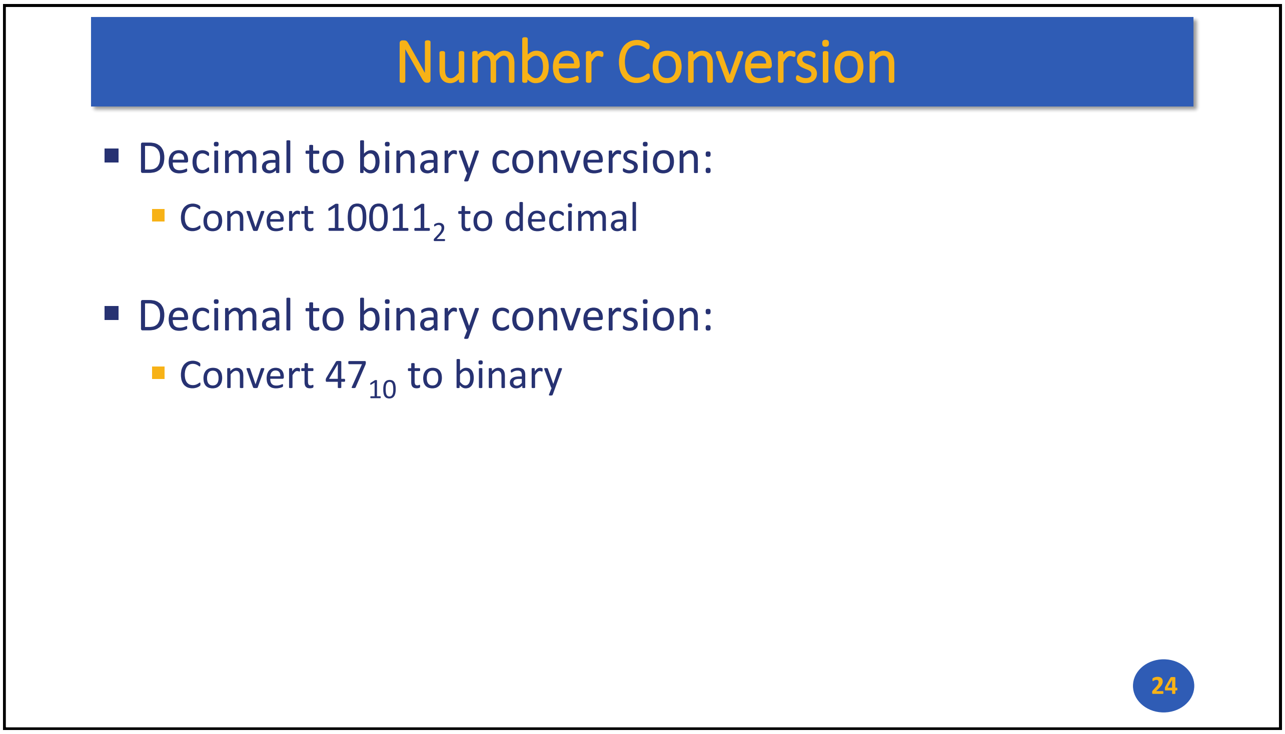
8's column
4's column
2's column
1's column

$$1101_2 = \underbrace{1 \times 2^3}_{\text{one eight}} + \underbrace{1 \times 2^2}_{\text{one four}} + \underbrace{0 \times 2^1}_{\text{no two}} + \underbrace{1 \times 2^0}_{\text{one one}} = 13_{10}$$



Powers of Two

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$



Number Conversion

- Decimal to binary conversion:
  - Convert 10011_2 to decimal
- Decimal to binary conversion:
  - Convert 47_{10} to binary

Binary Values and Range

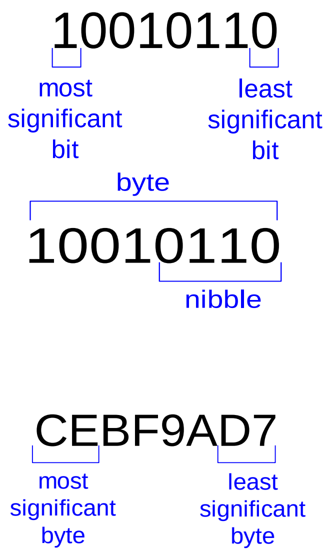
- *N*-digit decimal number
  - How many values? **10^N**
  - Range? **[0, 10^N - 1]**
  - Example: 3-digit decimal number:
    - **10^3 = 1000 possible values**
    - **Range: [0, 999]**
- *N*-bit binary number
  - How many values? **2^N**
  - Range: **[0, 2^N - 1]**
  - Example: 3-digit binary number:
    - **2^3 = 8 possible values**
    - **Range: [0, 7] = [000_2 to 111_2]**

Encoding Byte Values

- Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write $FA1D37B_{16}$ in C as
      - 0xFA1D37B
      - 0xfa1d37b

Bits, Bytes, Nibbles…

- Bits
- Bytes & Nibbles
- Bytes



27

Hexadecimal Numbers

- Base 16
- Shorthand for binary

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

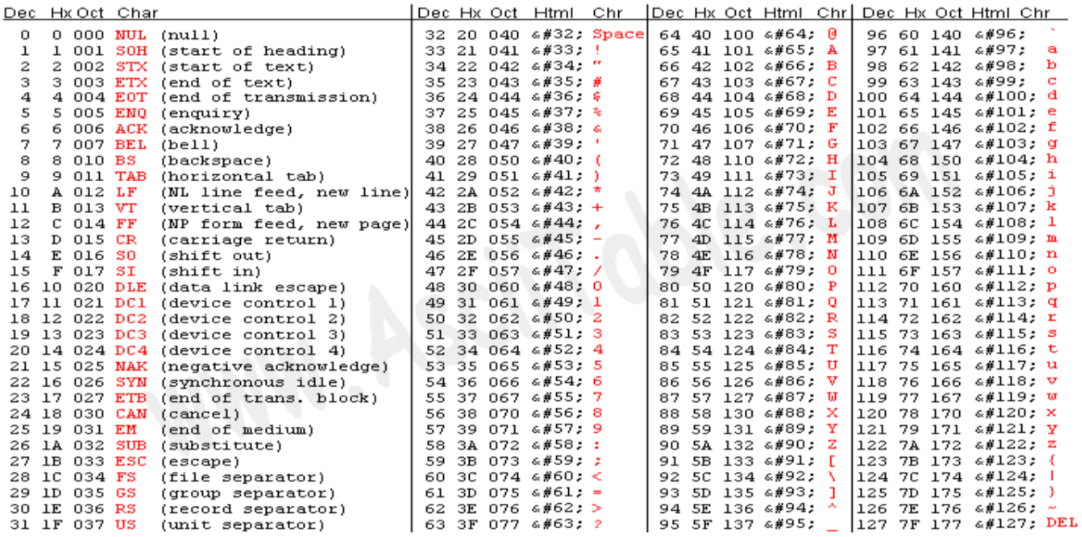
28

Hexadecimal to Binary Conversion

- Hexadecimal to binary conversion:
  - Convert $4AF_{16}$ (also written 0x4AF) to binary

- Hexadecimal to decimal conversion:
  - Convert $4AF_{16}$ to decimal

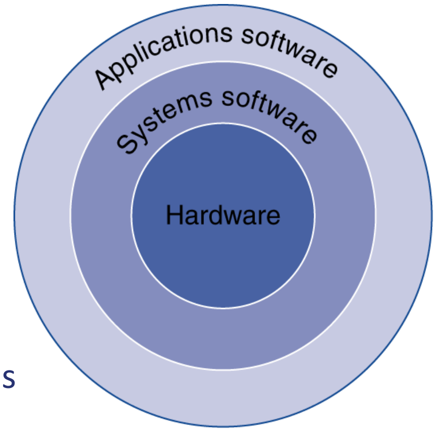
ASCII Code



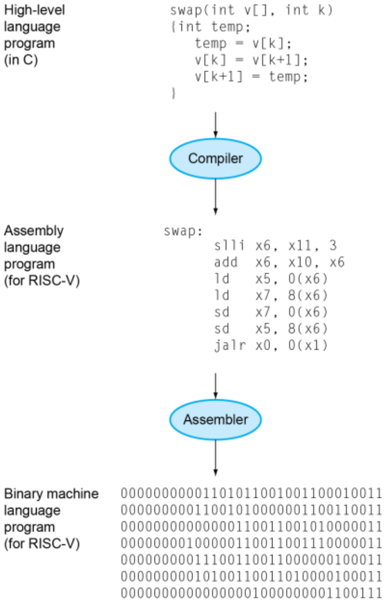
Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;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

Below Your Program

- Application software
  - Written in high-level language
- System software
  - Compiler: translates high-level language code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - CPU, memory, I/O controllers



Levels of Program Code

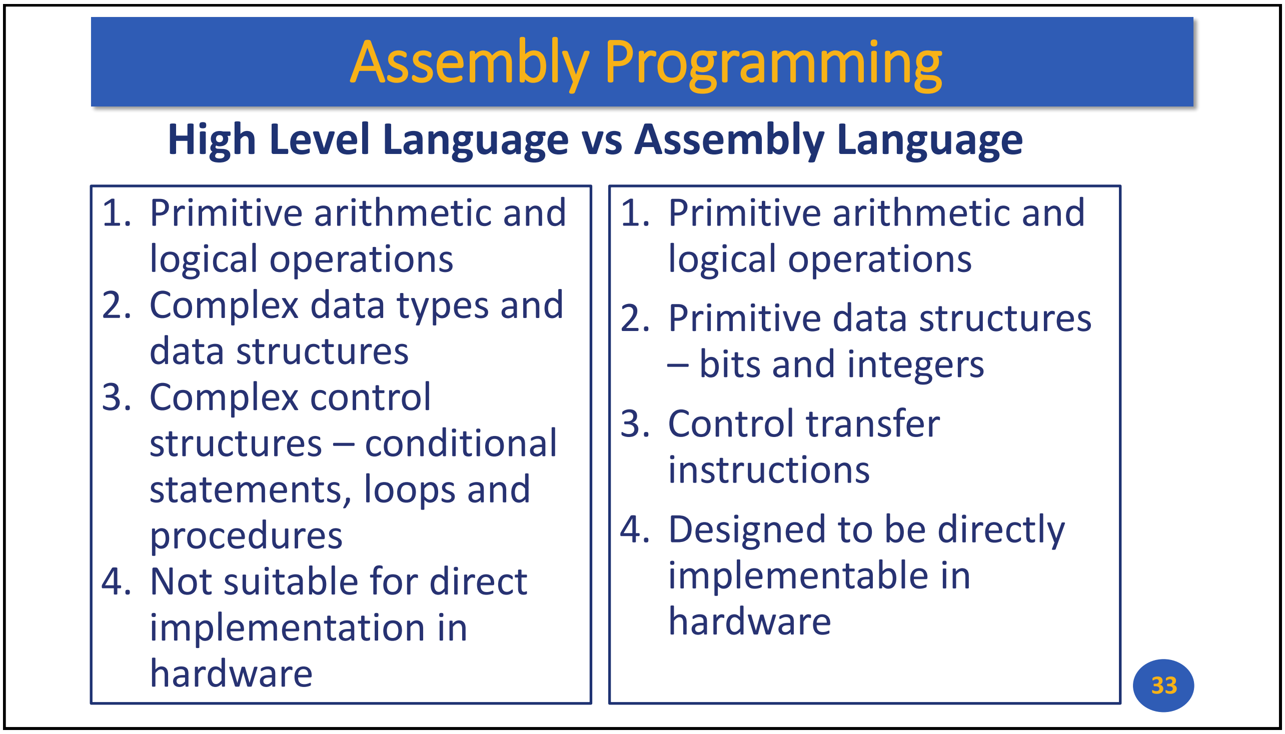

High-level language program (in C)
Compiler
Assembly language program (for RISC-V)
Assembler
Binary machine language program (for RISC-V)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      ld   x5, 0(x6)
      ld   x7, 8(x6)
      sd   x7, 0(x6)
      sd   x5, 8(x6)
      jalr x0, 0(x1)
```

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```

- High-level language
  - Level of abstraction closer to problem domain
  - Provides productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Assembly Programming

High Level Language vs Assembly Language

1. Primitive arithmetic and logical operations
2. Complex data types and data structures
3. Complex control structures – conditional statements, loops and procedures
4. Not suitable for direct implementation in hardware

1. Primitive arithmetic and logical operations
2. Primitive data structures – bits and integers
3. Control transfer instructions
4. Designed to be directly implementable in hardware