COMPUTER ARCHITECTURE

2nd Year Computer science

Chapiter1:

Introduction to computer architecture

Abdelhafid Boussouf University Center
2025-2026

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).

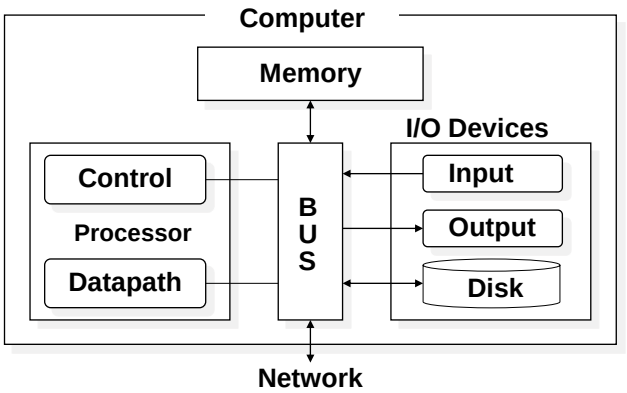
What Is Computer Architecture?

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

Components of a Computer System

- Processor
  - Datapath and Control
- Memory & Storage
  - Main Memory
  - Disk Storage
- Input / Output devices
  - User-interface devices
  - Network adapters
    - For communicating with other computers
- Bus: Interconnects processor to memory and I/O
- Essentially the same components for all kinds of computers



4

Classes of Computers

- Personal computers
  - General purpose, variety of software, subject to cost/performance
- Server computers
  - Network based, high capacity, performance, and reliability
  - Range from small servers to building sized
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but only a small fraction of the computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

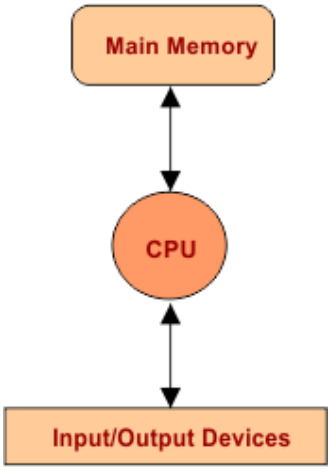
5

Classes of Computers

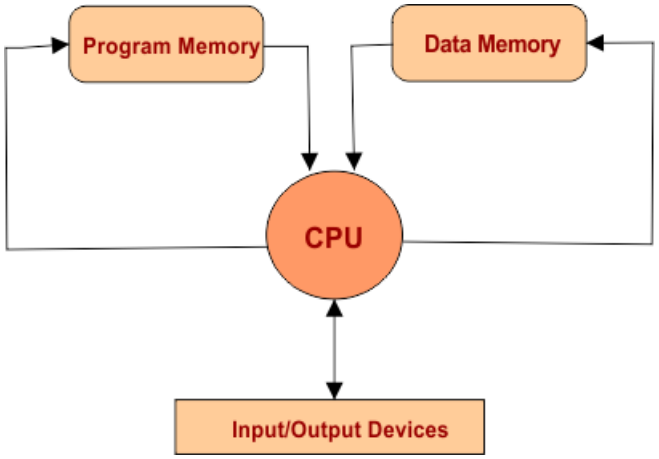
- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Low price: hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud Computing
  - Warehouse Scale Computers (WSC)
  - Software, Platform, and Infrastructure as a Service
  - However, security concerns of storing "sensitive data" in "the cloud"
  - Examples: Amazon and Google

6

Von Neumann Architecture and Harvard Architecture



Von Neumann Architecture

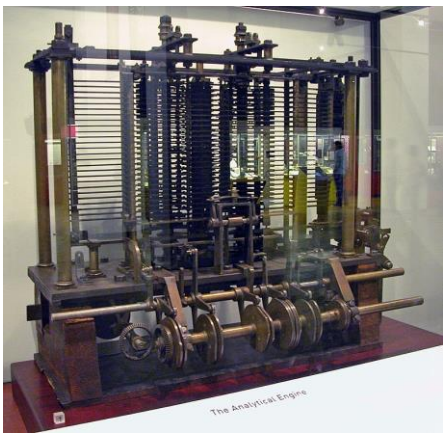


Harvard Architecture

7

History: 0th Generation – Mechanical

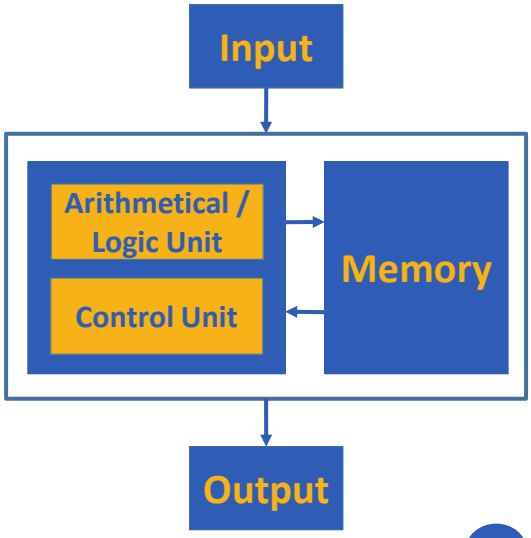
- 1834–71: Analytical Engine designed by Charles Babbage
- Mechanical gears, where each gear represented a discrete value (0-9)
- Programs provided as punched cards
- Never finished due to technological restrictions



8

History: 1st Generation - Vacuum Tubes

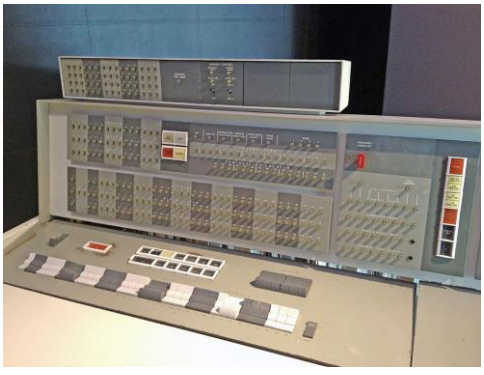
- 1945–55: first machines were created (Atanasoff–Berry, Z3, Colossus, ENIAC)
- All programming in pure machine language
- Connecting boards and wires, punched cards (later)
- Stored program concept



9

History: 2nd Generation - Transistors

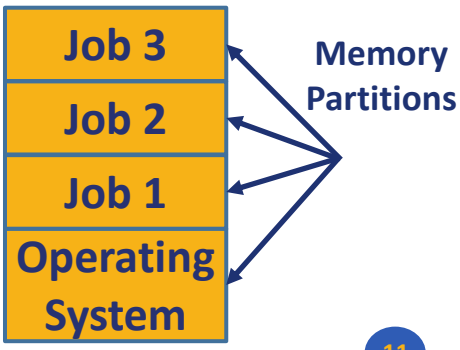
- 1955–65: era of mainframes (e.g. IBM 7094) used in large companies
- Programming in assembly language and FORTRAN
- Batch systems (IO was separated from calculations)
- Punched cards and magnetic tape
- Loaders (OS ancestors)



10

History: 3rd Generation – Integrated Circuits

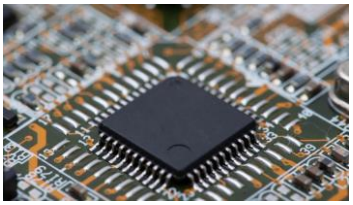
- 1965–1980: computer lines using the same instruction set architecture (e.g. IBM 360)
- First operating systems (e.g. OS/360, MULTICS)
- Multiprogramming and timesharing
- Computer as utility
- Programming languages and compilers (LISP, BASIC, C)



11

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



12

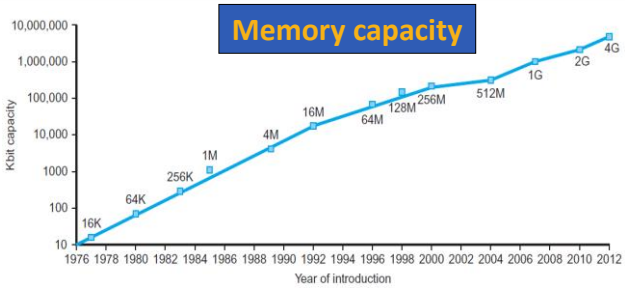
History: 5th Generation – Mobile devices

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

13

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

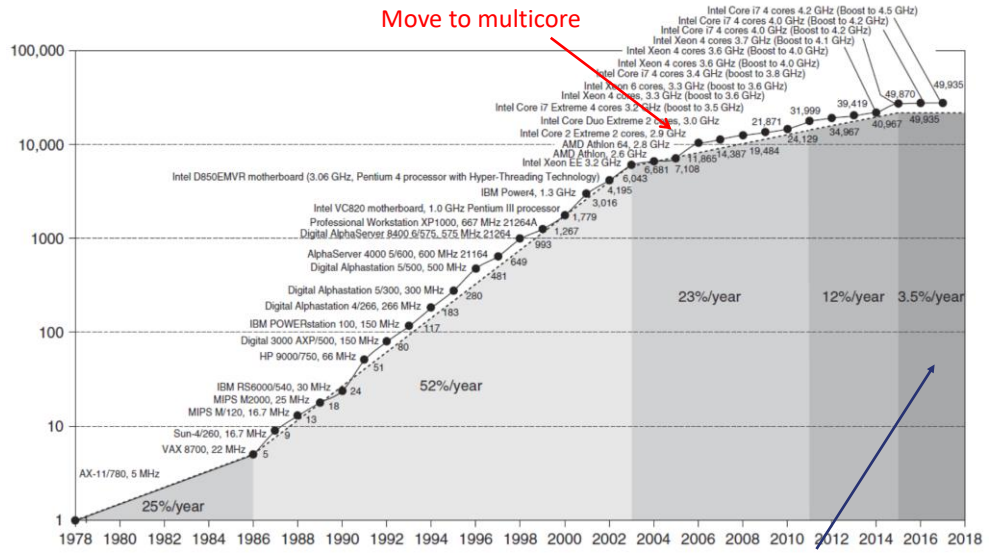
14

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

15

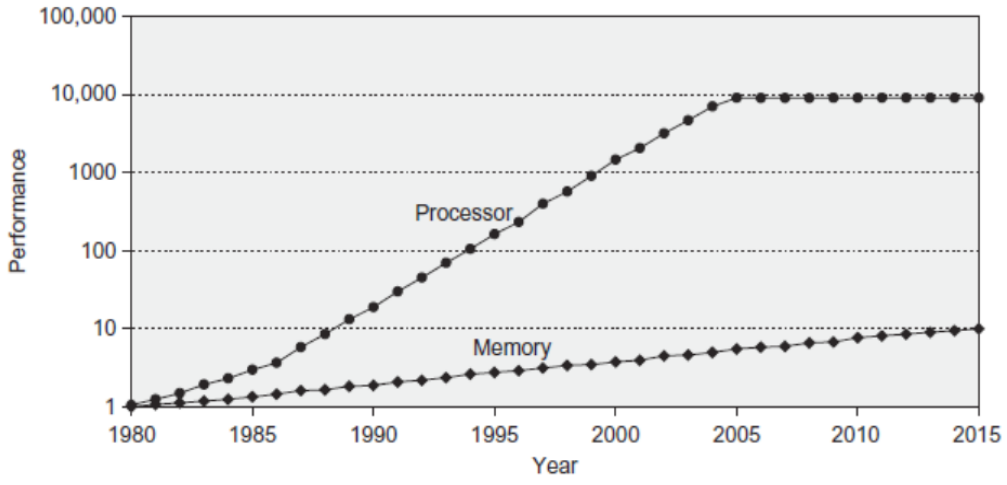
Single Core Performance



Constrained by power, instruction-level parallelism, memory latency

16

Memory Performance Gap



17

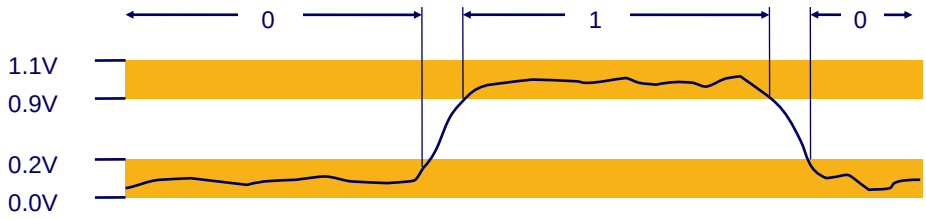
Current Challenges

- 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

18

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



19

Number Systems

- Decimal numbers

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

$$5374_{10} = 5 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$$

five thousands · three hundreds · seven tens · four ones

- Binary numbers

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

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

one eight · one four · no two · one one

20

Powers of Two

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

21

Number Conversion

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

22

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 $111_2]$

23

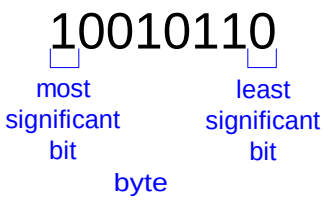
Encoding Byte Values

- 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

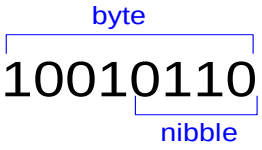
24

Bits, Bytes, Nibbles…

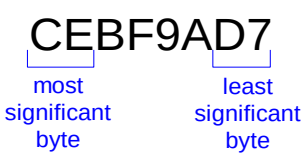
- Bits



- Bytes & Nibbles



- Bytes



25

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

26

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

27

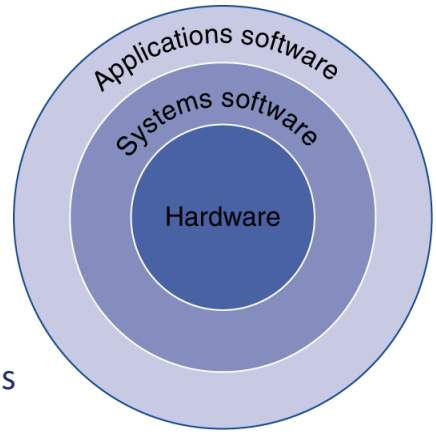
ASCII Code

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		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

28

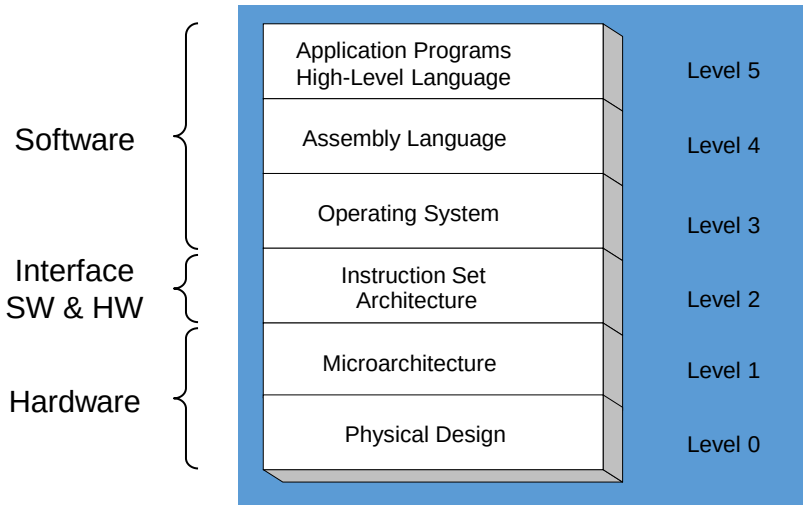
Below Your Program

- 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



29

Programmer's View of a Computer System



Increased level of abstraction

Each level hides the details of the level below it

30

Programmer's View of a Computer System

- **Application Programs (Level 5)**
  - Written in high-level programming languages
  - Such as Java, C++, Pascal, Visual Basic . . .
  - Programs compile into assembly language level (Level 4)
- **Assembly Language (Level 4)**
  - Instruction mnemonics (symbols) are used
  - Have one-to-one correspondence to machine language
  - Calls functions written at the operating system level (Level 3)
  - Programs are translated into machine language (Level 2)
- **Operating System (Level 3)**
  - Provides services to level 4 and 5 programs
  - Translated to run at the machine instruction level (Level 2)

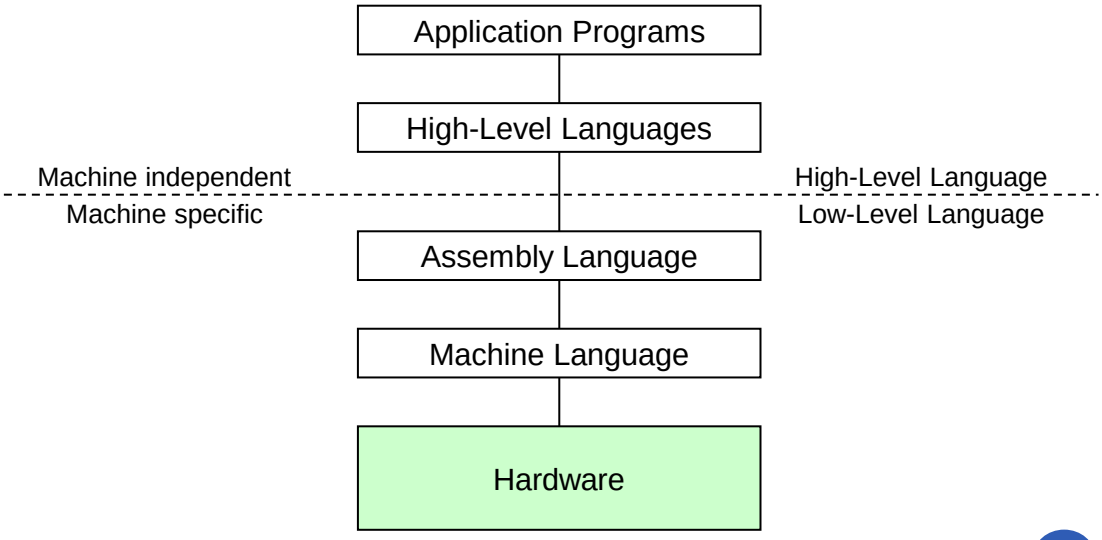
31

Programmer's View of a Computer System

- **Instruction Set Architecture (Level 2)**
  - Interface between software and hardware
  - Specifies how a processor functions
  - Machine instructions, registers, and memory are exposed
  - Machine language is executed by Level 1 (microarchitecture)
- **Microarchitecture (Level 1)**
  - Controls the execution of machine instructions (Level 2)
  - Implemented by digital logic
- **Physical Design (Level 0)**
  - Implements the microarchitecture at the transistor-level
  - Physical layout of circuits on a chip

32

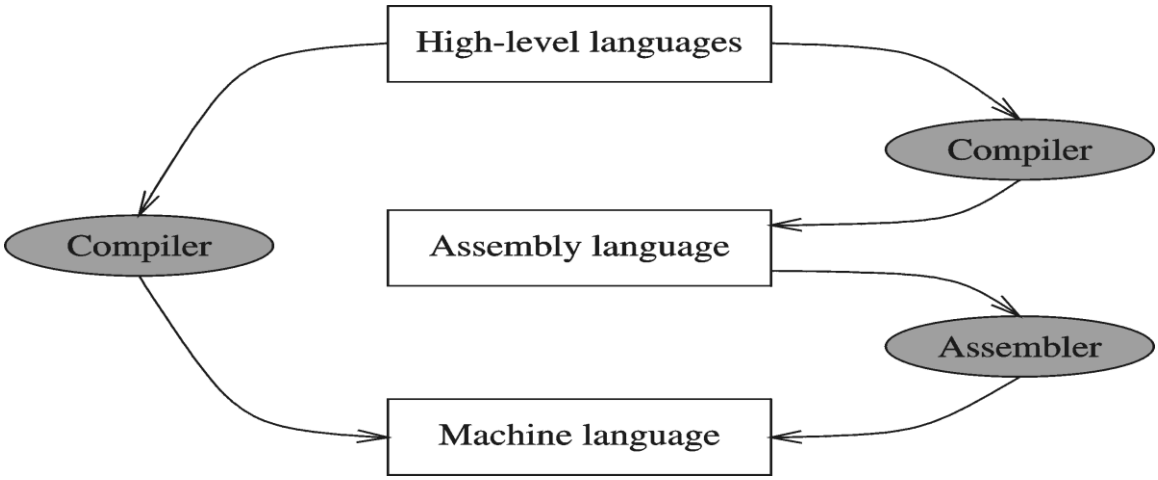
A Hierarchy of Languages



Assembly and Machine Language

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides productivity and portability
- **Machine language**
  - Native to a processor: executed directly by hardware
  - Instructions consist of binary code: 1s and 0s
- **Assembly language**
  - Slightly higher-level language
  - Readability of instructions is better than machine language
  - One-to-one correspondence with machine language instructions
- **Assemblers translate assembly to machine code**
- **Compilers translate high-level programs to machine code**
  - Either directly, or
  - Indirectly via an assembler

Compiler and Assembler



Translating Languages

Program (C Language):

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

Compiler

MIPS Assembly Language:

```
sll $2,$5, 2
add $2,$4,$2
lw  $15,0($2)
lw  $16,4($2)
sw  $16,0($2)
sw  $15,4($2)
jr  $31
```

Assembler

MIPS Machine Language:

```
00051080
00821020
8C620000
8CF20004
ACF20000
AC620004
03E00008
```

A statement in a high-level language is translated typically into several machine-level instructions

36

Advantages of High-Level Languages

- Program development is faster
  - High-level statements: fewer instructions to code
- Program maintenance is easier
  - For the same above reasons
- Programs are portable
  - Contain few machine-dependent details
    - Can be used with little or no modifications on different machines
  - Compiler translates to the target machine language
  - However, Assembly language programs are not portable

37

Why Learn Assembly Language?

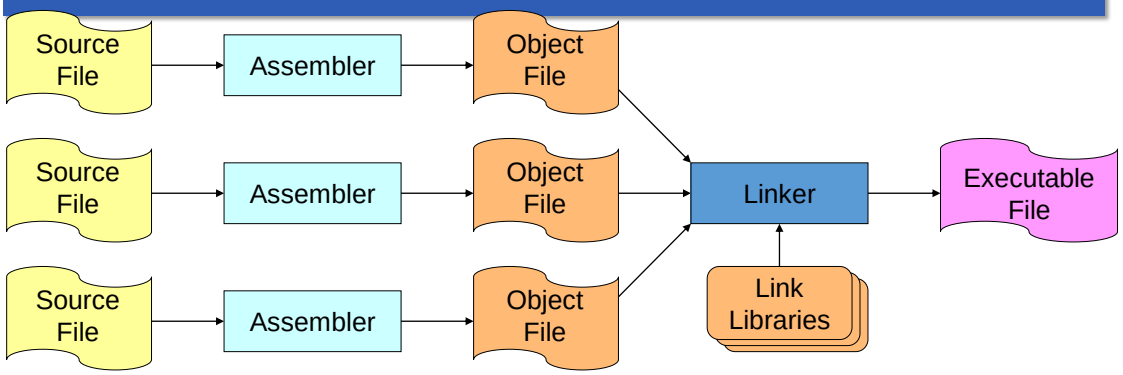
- **Many reasons:**
  - Accessibility to system hardware
  - Space and time efficiency
  - Writing a compiler for a high-level language
- **Accessibility to system hardware**
  - Assembly Language is useful for implementing system software
  - Also useful for small embedded system applications
- **Programming in Assembly Language is harder**
  - Requires deep understanding of the processor architecture
  - However, it is very rewarding to system software designers
  - Adds a new perspective on how programs run on real processors

38

Assembly Language Programming Tools

- **Editor**
  - Allows you to create and edit assembly language source files
- **Assembler**
  - Converts assembly language programs into object files
  - Object files contain the machine instructions
- **Linker**
  - Combines object files created by the assembler with link libraries
  - Produces a single executable program
- **Debugger**
  - Allows you to trace the execution of a program
  - Allows you to view machine instructions, memory, and registers

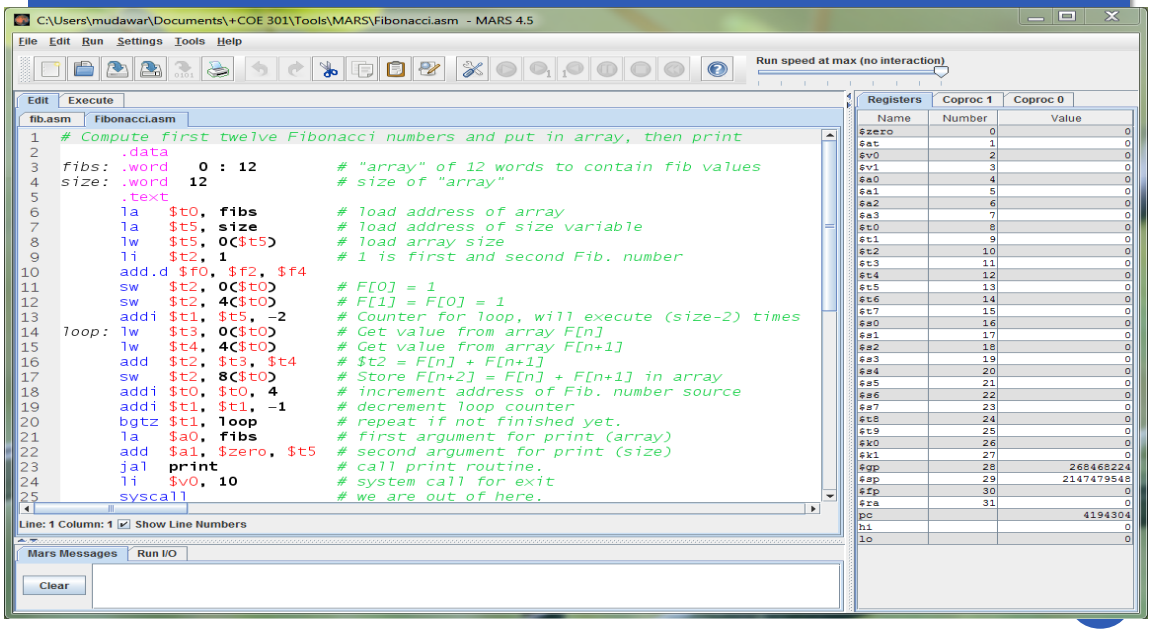
Assemble and Link Process



- A program may consist of multiple source files
- Assembler translates each source file into an object file
- Linker links all object files together and with link libraries
- The result executable file can run directly on the processor

40

MARS Assembler and Simulator Tool



MARS Assembler and Simulator Tool

- Simulates the execution of a MIPS program
  - No direct execution on the underlying Intel processor
- Editor with color-coded assembly syntax
  - Allows you to create and edit assembly language source files
- Assembler
  - Converts MIPS assembly language programs into object files
- Console and file input/output using system calls
- Debugger
  - Allows you to trace the execution of a program and set breakpoints
  - Allows you to view machine instructions, edit registers and memory
- Easy to use and learn assembly language programming

42