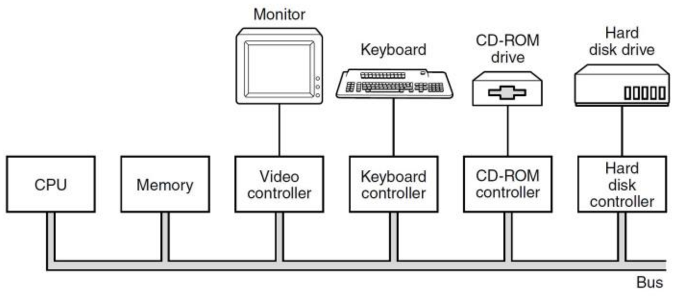
Peripheral Devices: I/O

We will examine some of these in the OS section



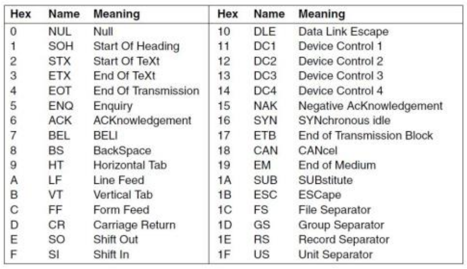
Buses, Terminals, Keyboards, Printers, Storage Devices (CD, DVD, HDD, Tape, …)

Drivers and Controllers (Speaking the "protocol" of the PC and the peripheral)

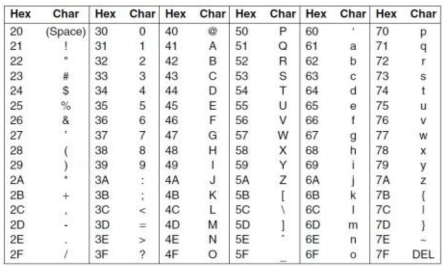
Character Encoding: ASCII, Unicode

83

Ascii Character Set



Hex	Name	Meaning	Hex	Name	Meaning
0	NUL	Null	10	DLE	Data Link Escape
1	SOH	Start Of Heading	11	DC1	Device Control 1
2	STX	Start Of TeXt	12	DC2	Device Control 2
3	ETX	End Of TeXt	13	DC3	Device Control 3
4	EOT	End Of Transmission	14	DC4	Device Control 4
5	ENQ	Enquiry	15	NAK	Negative AcKnowledgement
6	ACK	ACKnowledgement	16	SYN	SYNchronous idle
7	BEL	BELl	17	ETB	End of Transmission Block
8	BS	BackSpace	18	CAN	CANcel
9	HT	Horizontal Tab	19	EM	End of Medium
A	LF	Line Feed	1A	SUB	SUBstitute
B	VT	Vertical Tab	1B	ESC	ESCape
C	FF	Form Feed	1C	FS	File Separator
D	CR	Carriage Return	1D	GS	Group Separator
E	SO	Shift Out	1E	RS	Record Separator
F	SI	Shift In	1F	US	Unit Separator



Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20	(Space)	30	0	40	@	50	P	60	'	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	'	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

American System for Coded Information Interchange

84

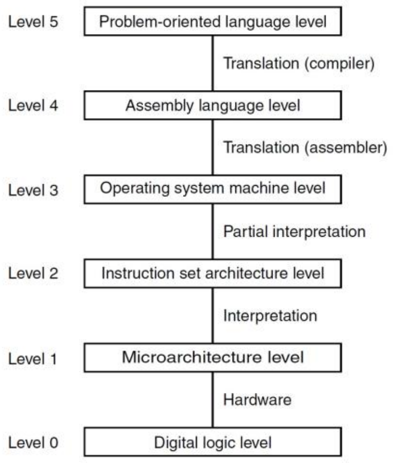
The Assembly Language Level

"High-Level" programs are translated to assembly language programs.

Translation is used when a processor (either hardware or an interpreter) is available for the target language but not for the source language.

There are two distinct steps:
1. Generation of an equivalent program.
2. Execution of the newly generated program.

Each assembly language statement corresponds to one machine instruction: there is a one-to-one correspondence between machine instructions and statements in the assembly program.



85

The Assembly Language Level

- Assembly language abstract away from the machine language by using symbolic names for instructions instead of binary or hexadecimal values:
  - ADD, SUB, MUL, and DIV,
- Symbolic names can also be used for memory locations
  - the assembler will generate correct numerical values.
- In contrast, a machine language programmer would have to work with the numerical values of operator and addresses.

86

The Assembly Language Level

- The assembly programmer has access to all the features and instructions available on the target machine. The high-level language programmer does not.
  - Eg: if the target machine has an overflow bit, an assembly language program can test it, but a Java program cannot.
- An assembly language program can execute every instruction in the instruction set of the target machine, but the high-level language program cannot.
- In short, everything that can be done in machine language can be done in assembly language, but many instructions, registers, and similar features are not directly available to the high-level language programmer.
- Languages for system programming, like C, are a cross between these types, with the syntax of a high-level language but with some of the access to the machine of an assembly language.
- An assembly language program can run only on one family of machines, whereas a program written in a high-level language can potentially run on many machines.

87

Why Program in Assembly?

- (1) Performance
  - An expert assembly language programmer can produce code that is much smaller and much faster than a high-level language programmer. For some applications, speed and size are critical. Many embedded applications, such as the code on a smart card or RFID card, device drivers, string manipulation libraries, BIOS routines, and the inner loops of performance-critical real-time applications fall in this category.

88

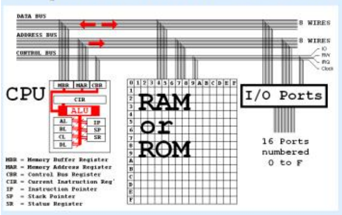
Why Program in Assembly?

- (2) Access to the machine.
  - Some procedures need complete access to the hardware, something usually impossible in high-level languages. For example, the low-level interrupt and trap handlers in an operating system and the device controllers in many embedded real-time systems fall into this category.
- (3) Understanding assembly language is essential to understanding how compilers work.
- (4) Writing some assembly code is the only way to get a feel for what a machine is really like at the architectural level.

89

Introducing Samphire

Simplified Simulator Architecture



Peripherals

- Keyboard Input
- Traffic Lights
- Seven Segment Display
- Heater and Thermostat
- Snake and Maze
- Stepper Motor
- Memory Mapped VDU

- central processing unit (CPU)
- 256 bytes of random access memory (RAM)
- 16 input output (IO) ports. Only six are used.
- A hardware timer that triggers interrupt 02 at regular time intervals that you can pre-set using the configuration tab.
- A keyboard that triggers interrupt 03.
- Peripherals connected to the Ports.

Description of the Simulator

The simulator includes a small sub-set of the full instruction set normally found with this style of processor. It includes advanced instructions such as CALL, RET, INT and IRET. There is a hardware timer interrupt simulation too.

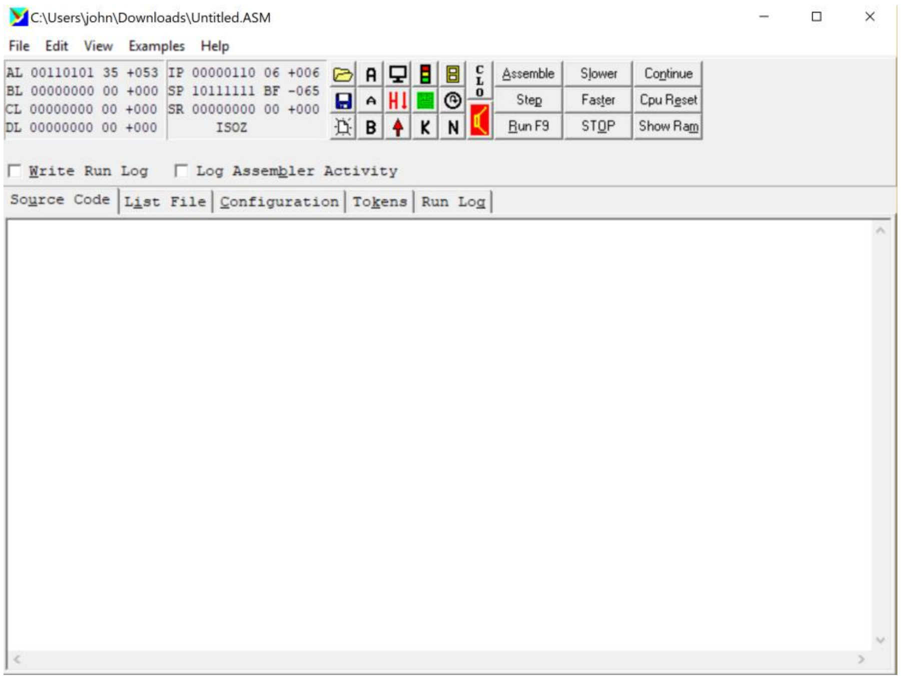
This simulator emulates an eight bit CPU that is similar to the low eight bits of the 80x86 family of chips. 256 bytes of RAM are simulated. It is surprising how much can be done with only 256 bytes or RAM.

The simulator is licensed under GNU / GPL making it freely available for use by students and educational institutions at zero cost.

Features

- 8 bit CPU
- 16 Input Output ports. Not all are used.
- Simulated peripherals on ports 0 to 5.
- An assembler.
- On-line help.
- Single step through programs.
- Continuously run programs.
- Interrupt 02 triggered by a hardware timer (simulated).
- CPU Clock Speed can be altered.

90



```
AL 00110101 35 +053 IP 00000110 06 +006
BL 00000000 00 +000 SP 10111111 BF -065
CL 00000000 00 +000 SR 00000000 00 +000
DL 00000000 00 +000        ISOZ
```

91