Assembly Language

Hashdump Security Club

High-level languages

- Provide convenient abstractions to help programmers
  - Variables, objects, if-else statements, loops, etc.
- But these are an abstraction: actual logic is more complex
- Your CPU doesn't understand C, Java, or Python on its own

```c
#include <stdio.h>

int main(int argc, char **argv) {
    puts("Hello World!");
    return 0;
}
```

Machine language & assembly

- Compilers/interpreters translate high-level languages into **machine code**
  - Short sequences of bytes
  - Fundamental operations supported by CPU
- **Assembly language:** human-readable text representation of machine code
  - Can be translated to machine language using an **assembler**
- Conversely, the code on the right was **disassembled** from a compiled binary
  - Will revisit this in a moment

```c
#include <stdio.h>

int main(int argc, char **argv) {
    puts("Hello World!");
    return 0;
}
```

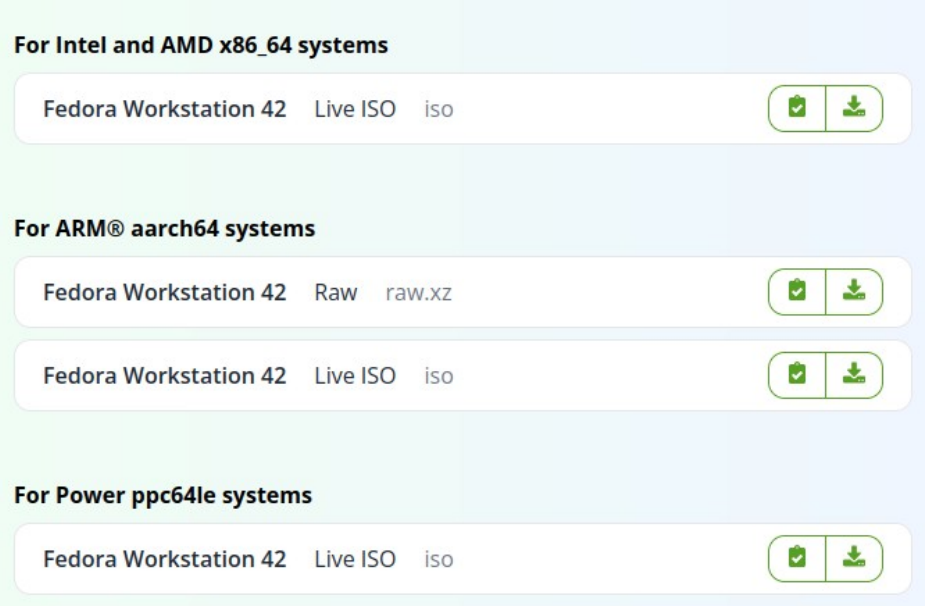
Ghidra disassembly of C “Hello World”

```
Address  Machine code     Assembly instructions

00101139 55               PUSH        RBP
0010113a 48 89 e5         MOV         RBP,RSP
0010113d 48 83 ec 10      SUB         RSP,0x10
00101141 89 7d fc         MOV         dword ptr [RBP + local_c],EDI
00101144 48 89 75 f0      MOV         qword ptr [RBP + local_18],RSI
00101148 48 8d 05         LEA         RAX,[s_Hello_World!_00102004]
         b5 0e 00 00
0010114f 48 89 c7         MOV         RDI=>s_Hello_World!_00102004,RAX
00101152 e8 d9 fe         CALL        <EXTERNAL>::puts
         ff ff
00101157 b8 00 00         MOV         EAX,0x0
         00 00
0010115c c9               LEAVE
0010115d c3               RET
```

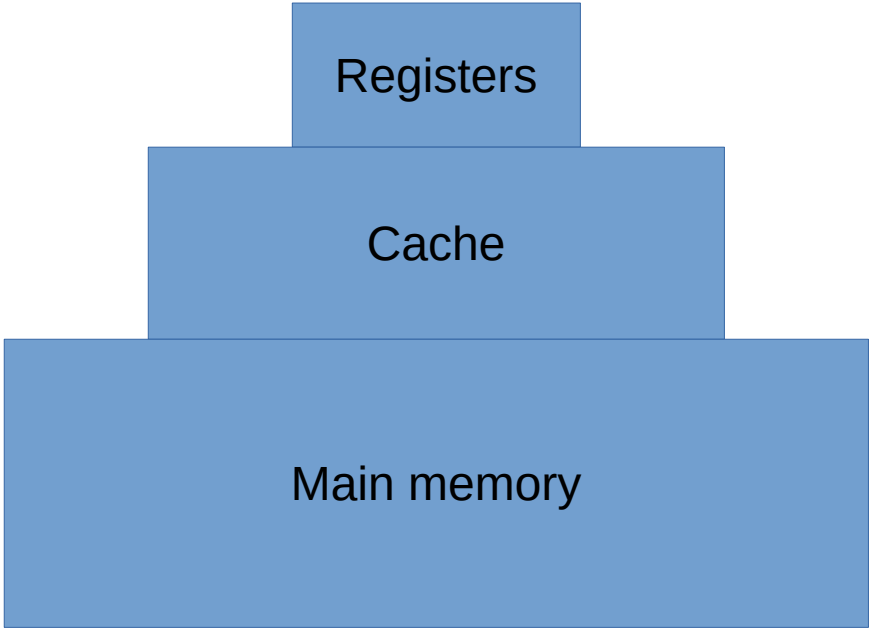
Instruction set architecture (ISA)

- Different CPUs support different instructions
- Most desktops/laptops use x86_64
- Another common architecture: ARM
  - Phones, MacBooks, Raspberry Pis



Memory hierarchy

- **Main memory:** Random-access memory (RAM)
- **Cache:** Recently/frequently-used memory
- **Registers:** Data actively being used
  - Instructions often operate directly on registers
- Unlike RAM, registers and cache are **part of the CPU itself**
  - Caching compensates for RAM latency, but registers are still the fastest
  - However, registers are also smallest in size



Instruction set example

- MOS Technology 6502 processor
- Used on NES, Apple II, and Atari 2600, among others
- Considerably smaller instruction set as compared to x86

Register	***Description***
Program counter	How far the CPU is along the program
X, Y	General-purpose registers
A	Accumulator (more math/binary operations)

Example instruction	***Description***
LDA, LDX, LDY	Load a value from RAM into register
STA, STX, STY	Store a value from a register into RAM
INX, INY, DEX, DEY	Increment/decrement register
JMP	Jump (move the program counter) to a different part of the program

Revisiting x86 Hello World

```
PUSH    RBP                                  ; Push a new stack frame
MOV     RBP,RSP
SUB     RSP,0x10                             ; Allow space for local variables on stack
MOV     dword ptr [RBP + local_c],EDI        ; Load function arguments onto stack
MOV     qword ptr [RBP + local_18],RSI
LEA     RAX,[s_Hello_World!_00102004]        ; Load address of string
MOV     RDI=>s_Hello_World!_00102004,RAX     ; Prepare argument for puts call
CALL    <EXTERNAL>::puts                     ; Invoke print routine
MOV     EAX,0x0                              ; Set return value
LEAVE                                        ; Return from function
RET
```

Instruction	***Args***	***Description***
PUSH	*val*	Push onto stack
MOV	*dst,src*	Move (copy) value
SUB	*loc amt*	Subtract
LEA	*dst,val*	Load Effective Address
CALL	*proc*	Call Procedure
LEAVE		High Level Procedure Exit
RET		Return From Procedure

Register		***Description***
64-bit	***32-bit***	
RBP	EBP	Stack Base Pointer
RSP	ESP	Stack Pointer
RDI	EDI	Destination
RSI	ESI	Source
RAX	EAX	Accumulator

Note: This is a disassembly of C code, not how you would implement Hello World in pure ASM.

Plain x86 (no C library)

```
; https://www.learningaboutelectronics.com/Articles/Hello-world-in-x86-assembly.php

org 100h

jmp main

message: db 'Hello World!', 0

print:
    mov ah, 0eh     ; Use teletype output
._loop:
    lodsb           ; Read character from the string
    cmp al,0        ; Exit if we've reached the end (null byte)
    je .done
    int 10h         ; Write character to the terminal
    jmp ._loop      ; Repeat
.done:
    ret             ; Exit print function
main:
    mov si, message ; Call the print function, passing in message as argument
    call print
    ret             ; Print has exited, so exit the program
```

References

[1] J. Pickens, B. Clark, and E. Spittles, “6502.org: NMOS 6502 Opcodes.” Accessed: Sept. 07, 2025. [Online]. Available: http://www.6502.org/tutorials/6502opcodes.html

[2] N. Animal, “Answer to ‘What is the purpose of the RBP register in x86_64 assembler?,’” Stack Overflow. Accessed: Sept. 07, 2025. [Online]. Available: https://stackoverflow.com/a/41914096

[3] “CPU Registers x86 - OSDev Wiki.” Accessed: Sept. 07, 2025. [Online]. Available: https://wiki.osdev.org/CPU_Registers_x86

[4] “Hello World Program in x86 Assembly Language.” Accessed: Sept. 07, 2025. [Online]. Available: https://www.learningaboutelectronics.com/Articles/Hello-world-in-x86-assembly.php

[5] “INT 10H,” Wikipedia. June 19, 2025. Accessed: Sept. 07, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=INT_10H&oldid=1296382288

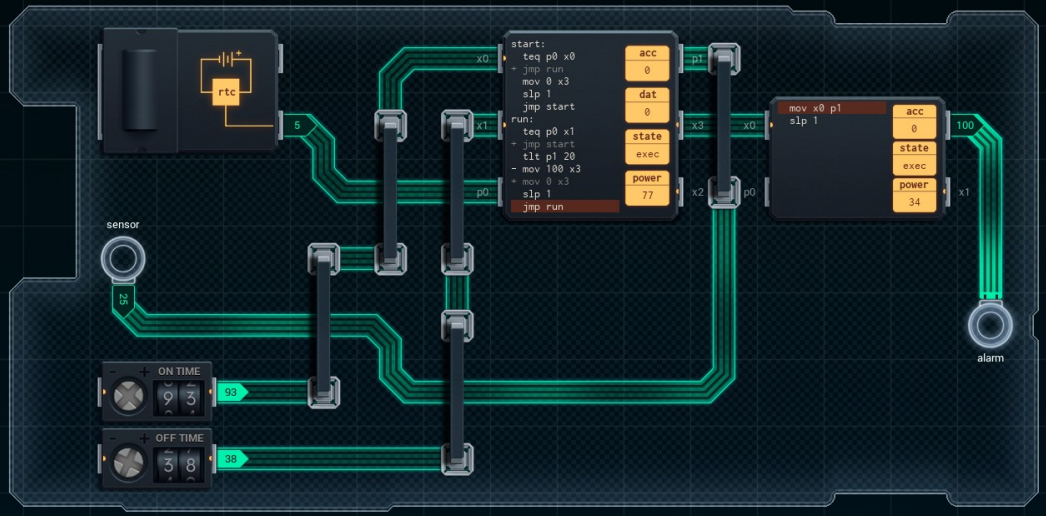
[6] “MOS Technology 6502,” Wikipedia. Sept. 04, 2025. Accessed: Sept. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/MOS_Technology_6502

[7] “Nintendo Entertainment System,” Wikipedia. Sept. 02, 2025. Accessed: Sept. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Nintendo_Entertainment_System

[8] J. Stokes, “Understanding CPU caching and performance,” Ars Technica. Accessed: Sept. 07, 2025. [Online]. Available: https://arstechnica.com/gadgets/2002/07/caching/

[9] “x86 and amd64 instruction reference.” Accessed: Sept. 07, 2025. [Online]. Available: https://www.felixcloutier.com/x86/

Activity: *SHENZHEN I/O*



SHOW WIRES TAB | TEST RUN 1/80 | PRODUCT COST ¥8 | POWER USAGE 111 | LINES OF CODE 14 | SHOW PROFILER R

RESET PAUSE STEP ADVANCE SIMULATE

INFORMATION VERIFICATION

time simple input
sensor simple input
alarm simple output

NOTE ¥0
BRIDGE ¥0
MC4000 ¥3
MC4000X ¥3
MC6000 ¥5
DX300 ¥1
100P-14 ¥2
200P-14 ¥2
LC70G04 ¥1