

12 – ASSEMBLY LANGUAGE

COMP256 – COMPUTING ABSTRACTIONS

DICKINSON COLLEGE

ASSEMBLY LANGUAGE PROGRAMS

- An assembly language is an abstraction for a machine language.

- Assembly language instructions:
 - Are mnemonic (human readable, easier to remember text) version of machine language instructions.
 - Usually have a 1-to-1 correspondence to machine language instructions.

Machine Language	Meaning	Assembly Language
1000 0001 0 10 00101	R2 = MM[5]	LOAD R2 5
1010 0001 00 11 00 01	R3 = R0 + R1	ADD R3 R0 R1

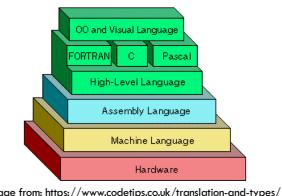


Image from: <https://www.codertips.co.uk/translation-and-types/>

ASSEMBLY LANGUAGE PROGRAMMING

- A thought before we begin...

"He who hasn't hacked assembly language as a youth has no heart.
He who does as an adult has no brain."

John Moore

ASSEMBLY LANGUAGES

- Assembly Languages are low level translated languages.
 - A program called an assembler translates the assembly source code program into an equivalent executable machine language program for the target machine.
 - Assembly language source code is specific to the target machine.
- Used in applications that require:
 - Precise control of the system
 - Highly optimized performance
 - E.g. Operating systems, device drivers, real-time control systems, game engines

Intel x86, IA-32, x86-64
ARM, MIPS, SPARC, DEC, IBM360,
Motorola 6800, etc ...

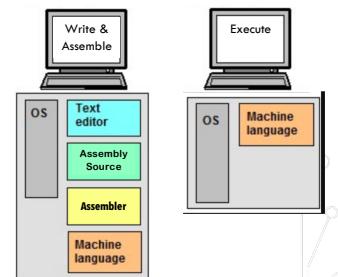
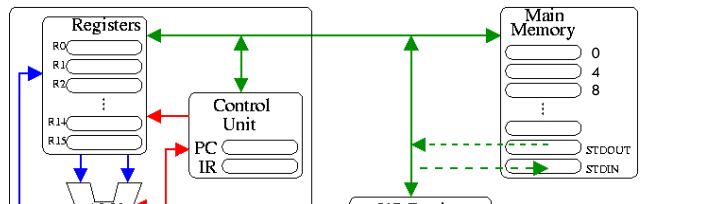


Image adapted from: <https://encyclopedia2.thefreedictionary.com/interpreter>

OUR ASSEMBLY LANGUAGE: THE MACHINE

- Simplified MIPS machine and assembly language



- 16 Registers (R0-R15)
 - 32 bits each
 - R0-R11 General Purpose
 - R12-R15 Reserved

- Two special memory addresses
 - STDOUT
 - STORE to here to display to console.
 - STDIN
 - LOAD from here to read from user.

OUR ASSEMBLY LANGUAGE: FIRST EXAMPLE

// HLL

```
int A=10;
int B=20;
int C;
```

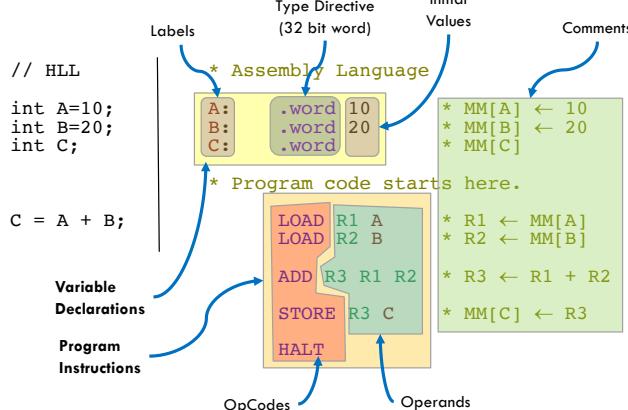
* Assembly Language

```
A: .word 10          * MM[A] ← 10
B: .word 20          * MM[B] ← 20
C: .word
```

* Program code starts here.

```
LOAD R1 A           * R1 ← MM[A]
LOAD R2 B           * R2 ← MM[B]
ADD R3 R1 R2        * R3 ← R1 + R2
STORE R3 C          * MM[C] ← R3
HALT
```

OUR ASSEMBLY LANGUAGE: TERMINOLOGY



OUR ASSEMBLY LANGUAGE: ASSEMBLING

- Make COMP256Machine folder
- Download Assembler.jar to COMP256Machine folder
- In terminal cd COMP256Machine folder
- Assemble source code program to machine language program
 - Syntax: java -jar Assembler.jar <assembly> <executable>
 - <assembly>: Filename of assembly language source code program to assemble.
 - <executable>: Filename for executable machine language program to create.
 - Example: java -jar Assembler.jar example1.asm example1.exe
 - Assembles the assembly language program in the file example1.asm into machine language and stores the result in the file example1.exe.
 - Note: example1.asm must be in the same folder as Assembler.jar

OUR ASSEMBLY LANGUAGE: RUNNING

- Download Machine.jar to COMP256Machine folder
- In terminal cd COMP256Machine folder
- Run machine language program
 - Syntax: `java -jar Machine.jar <executable>`
 - `<executable>`: Filename for machine language program to run.
- Example: `java -jar Machine.jar example1.exe`
 - Loads the machine language program in the file `example1.exe` into the machine simulator.
 - Note: `example1.exe` must be in the same folder as `Machine.jar`.

DIRECT ADDRESSING MODE INSTRUCTIONS

- In a **direct addressing mode** instruction the label is used (directly) as a memory address.
 - A label is a mnemonic name for a memory address assigned by the assembler.

Instruction	Example	Meaning	Comments
LOAD R L	LOAD R1 X	$R1 \leftarrow MM[X]$	Direct Mode
STORE R L	STORE R1 X	$MM[X] \leftarrow R1$	Direct Mode

REGISTER TO REGISTER INSTRUCTIONS

- A **register to register instruction** operates on values from the registers and stores the result into a register.

Instruction	Example	Meaning	Comments
ADD R R R	ADD R1 R2 R3	$R1 \leftarrow R2 + R3$	
SUB R R R	SUB R1 R2 R3	$R1 \leftarrow R2 - R3$	
AND R R R	AND R1 R2 R3	$R1 \leftarrow R2 \& R3$	Bitwise AND
OR R R R	OR R1 R2 R3	$R1 \leftarrow R2 R3$	Bitwise OR
NOT R R	NOT R1 R2	$R1 \leftarrow \sim R2$	Bitwise NOT
SHL R R	SHL R1 R2	$R1 \leftarrow R2 \ll 1$	LSb = 0
SHR R R	SHR R1 R2	$R1 \leftarrow R2 \ggg 1$	MSb = 0
MOV R R	MOV R1 R2	$R1 \leftarrow R2$	Copy

IMMEDIATE ADDRESSING MODE INSTRUCTIONS

- In an **immediate addressing mode** instruction the value of the operand preceded by a # is used as is (i.e. "immediately").
 - The value of an immediate mode operand can be given in decimal, hexadecimal or binary.

Instruction	Example	Meaning	Comments
ADD R R #	ADD R1 R2 #231	$R1 \leftarrow R2 + 231$	Immediate Mode
SUB R R #	SUB R1 R2 #1	$R1 \leftarrow R2 - 1$	Immediate Mode
AND R R #	AND R1 R2 #0xF00F	$R1 \leftarrow R2 \& 0xF00F$	Immediate Mode
OR R R #	OR R1 R2 #b1001	$R1 \leftarrow R2 b1001$	Immediate Mode
LOAD R #	LOAD R1 #27	$R1 \leftarrow 27$	Immediate Mode

OUR ASSEMBLY LANGUAGE: INPUT / OUTPUT

```
// HLL
int X;
int Z;

* Assembly Language
X: .word
Z: .word

* Program code starts here.

Read X;           LOAD R1 STDIN      * R1 ← MM[STDIN]
                  * MM[X] ← R1
Z = 2*X-1;       ADD R2 R1 R1    * R2 ← 2* R1
                  SUB R2 R2 #1     * R2 ← R2 - 1
                  STORE R2 Z      * MM[Z] ← R2
Print Z;          STORE R2 STDOUT   * MM[STDOUT] ← R2
                  HALT
```

ACTIVITY

- Download the Assembler and Machine from the course home page.
- Type the First Example in to a text editor.
 - Save it in the same directory as the Assembler and Machine.
- Assemble the First Example using the Assembler.
- Run the First Example using the Machine.

* Assembly Language

```
A: .word 10      * MM[A] ← 10
B: .word 20      * MM[B] ← 20
C: .word         * MM[C]

* Program code starts here.

LOAD R1 A        * R1 ← MM[A]
LOAD R2 B        * R2 ← MM[B]
ADD R3 R1 R2    * R3 ← R1 + R2
STORE R3 C       * MM[C] ← R3
HALT
```

ACTIVITY

- Write assemble and execute an assembly language program that accomplishes the same task as the following HLL program:

```
int A;
int B;
int C;

Read A;
Read B;

C = 2*(A+10) - (B-20);

Print C;
```