Assembly

How to share data across web page?

Language

ASSEMBLY LANGUAGE

Assembly language, or **assembly** or **ASM** is a low-level programming language for a computer or other programmable device specific to a particular **computer architecture** in contrast to most high-level programming languages, which are generally portable across multiple systems.

The stack machine

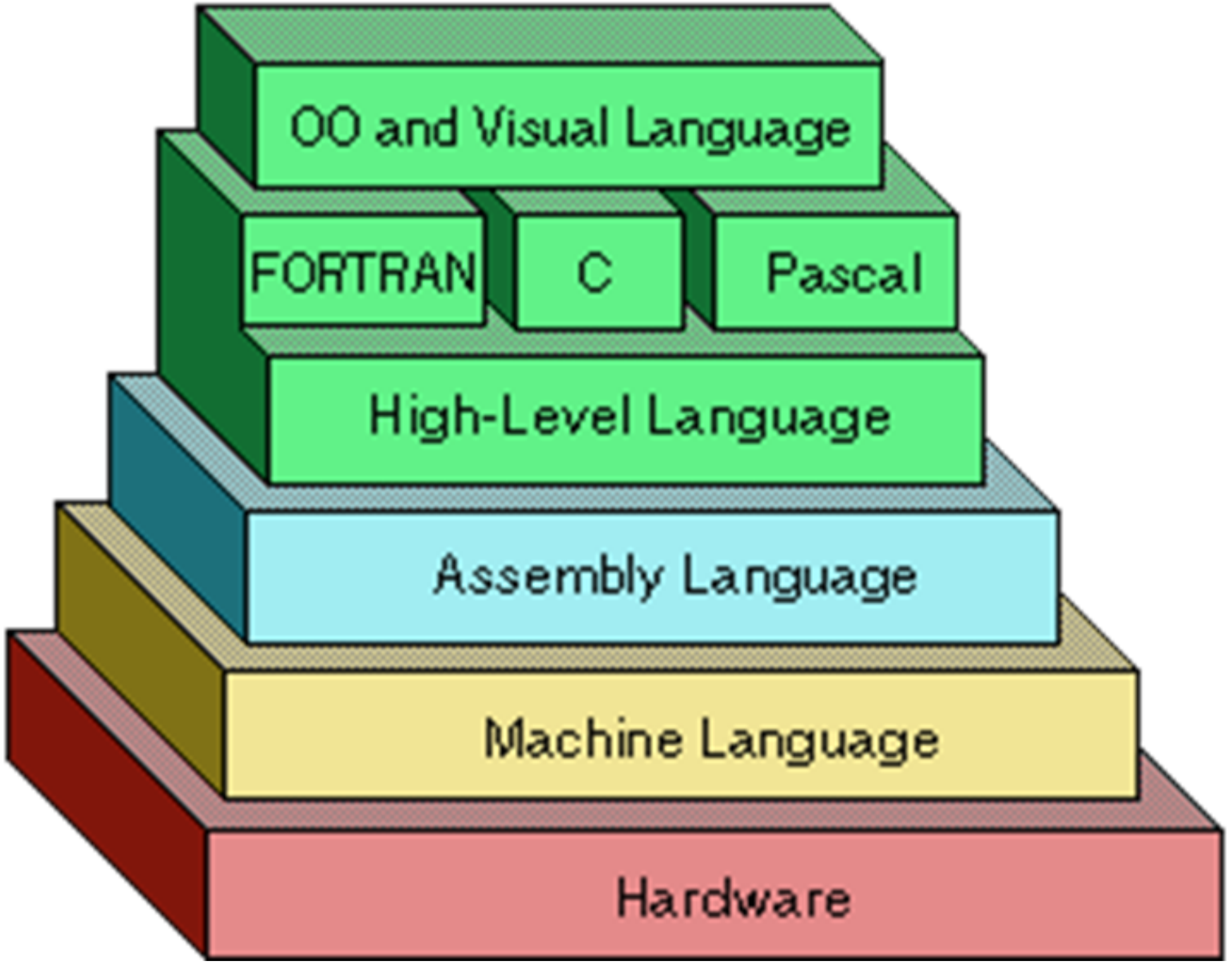
The accumulator machine

The load/store machine

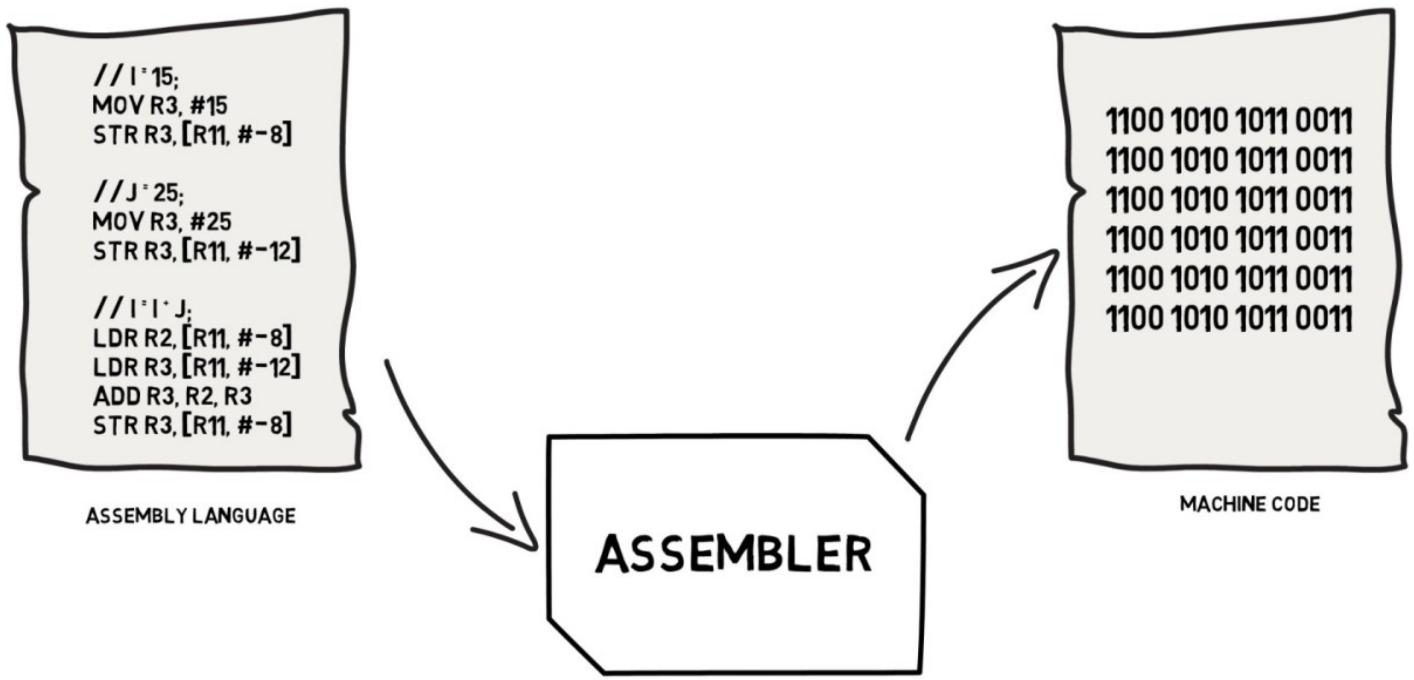
The most common computer architecture

http://faculty.salina.k-state.edu/tim/ossg/Assembly/machine_types.html

ASSEMBLY LANGUAGE



ASSEMBLY LANGUAGE



```
// I = 15;
MOV R3, #15
STR R3, [R11, #-8]

// J = 25;
MOV R3, #25
STR R3, [R11, #-12]

// I = I + J;
LDR R2, [R11, #-8]
LDR R3, [R11, #-12]
ADD R3, R2, R3
STR R3, [R11, #-8]
```

```
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
```

What are the advantages of learning assembly language?

Having an understanding of assembly language makes one aware of –

- How programs interface with OS, processor, and BIOS;
- How data is represented in memory and other external devices;
- How the processor accesses and executes instruction;
- How instructions access and process data;
- How a program accesses external devices.

Other advantages of using assembly language are –

- It requires less memory and execution time
- It allows hardware-specific complex jobs in an easier way
- It is suitable for time-critical jobs
- It is most suitable for writing interrupt service routines and other memory resident programs.

Assembly is different from one computer architecture to another.

So be aware that the code syntax you found later might be different from you write in the future.

X86 Assembly instructions

https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

ARM Assembly instructions

https://azeria-labs.com/writing-arm-assembly-part-1/

What are the advantages of learning assembly language?

Having an understanding of assembly language makes one aware of –

- How programs interface with OS, processor, and BIOS;
- How data is represented in memory and other external devices;
- How the processor accesses and executes instruction;
- How instructions access and process data;
- How a program accesses external devices.

Other advantages of using assembly language are –

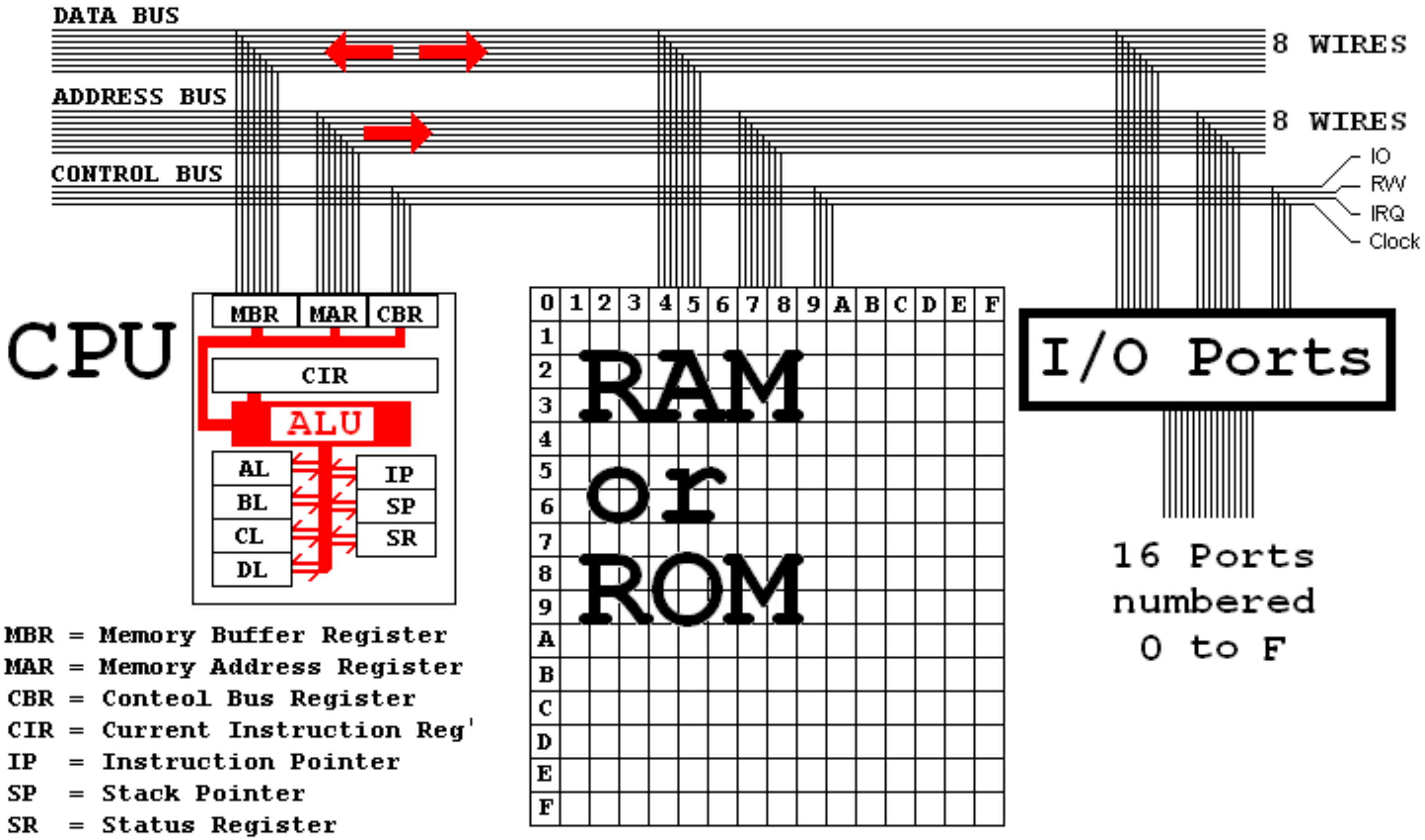
- It requires less memory and execution time
- It allows hardware-specific complex jobs in an easier way
- It is suitable for time-critical jobs
- It is most suitable for writing interrupt service routines and other memory resident programs.

ASSEMBLY LANGUAGE

The Microprocessor Simulator

http://www.softwareforeducation.com/sms32v50/index.php

The Microprocessor Simulator: The architecture



MBR = Memory Buffer Register
MAR = Memory Address Register
CBR = Conteol Bus Register
CIR = Current Instruction Reg'
IP = Instruction Pointer
SP = Stack Pointer
SR = Status Register

ASSEMBLY LANGUAGE

The Miscellaneous Operation

These are the instructions to initial or interrupt the flow of application.

The Procedure & Interrupts Operators

Assembler		Explanation
CLO		Close visible peripheral windows.
HALT		Halt the processor.
NOP		Do nothing for one clock cycle.
STI		Set the interrupt flag in the Status Register.
CLI		Clear the interrupt flag in the Status Register.
ORG	40	Assembler directive: Generate code starting from address 40.
DB	"Hello"	Assembler directive: Store the ASCII codes of 'Hello' into RAM.
DB	84	Assembler directive: Store 84 into RAM.

The Arithmetic & Logic Operation

Arithmetical operations are work with numbers while logical operations are work with boolean value.

The Arithmetic Operators

Addition Operator		
Direct addressing:	ADD R1, R2	R1 = R1 + R2
Immediate addressing:	ADD R, 12	R = R + 12

Substraction Operator		
Direct addressing:	SUB R1, R2	R1 = R1 - R2
Immediate addressing:	SUB R, 03	R = R - 03

Multiplication Operator		
Direct addressing:	MUL R1, R2	R1 = R1 * R2
Immediate addressing:	MUL R, 05	R = R * 05

Division Operator		
Direct addressing:	DIV R1, R2	R1 = R1 / R2
Immediate addressing:	DIV R, 02	R = R / 02

R, R1, R2 : register
12, 03, 02, 05 : value in hexadecimal

The Arithmetic Operators

Increase Operator		
Direct addressing:	INC R	R = R + 1

Decrease Operator		
Direct addressing:	DEC R	R = R -1

R, R1, R2 : register
12, 03, 02, 05 : value in hexadecimal

ASSEMBLY LANGUAGE

The Logic Operators

AND	AL,BL	AL	=	AL AND BL
OR	CL,BL	CL	=	CL OR BL
XOR	AL,BL	AL	=	AL XOR BL
NOT	BL	BL	=	NOT BL
ROL	AL	Rotate bits left. LSB = MSB		
ROR	BL	Rotate bits right. MSB = LSB		
SHL	CL	Shift bits left. Discard MSB.		
SHR	DL	Shift bits right. Discaed LSB.		

AL, BL, CL, DL : register
01, 02, 03, 00 : value in hexadecimal

The Data Move Operation

The instructions to move data around inside computer.

The Data Move Operators

Data Move Operator		
Immediate value to Register:	MOV R, 15	R = 15
RAM to Register:	MOV R, [12]	R = value of RAM at address 12
	MOV R1,[R2]	R1 = value of RAM at address (value of R2)
Register to RAM:	MOV [12], R	RAM at address 12 = R
	MOV [R1], R2	RAM at address (value of R1) = R2

You can’t move data from register to register. Either move to memory first, or use stack

R1, R2, R : register
12, 15 : value in hexadecimal

The Input Output Operation

The instructions to accept input from external devices such as keyboard, scanner.. while output instructions are used to display information.

The Procedure & Interrupts Operators

Assembler		**Explanation**
IN	7	Data input from I/O port 07 to AL.
OUT	1	Data output to I/O port 01 from AL.

The Stack Manipulation Operation

Inside primary memory, there is a stack which work as FIFO. Data can put inside the stack or move it out.

The Stack Manipulation Operators

Assembler		Explanation
PUSH	BL	BL is saved onto the stack.
POP	CL	CL is restored from the stack.
PUSHF		SR flags are saved onto the stack.
POPF		SR flags are restored from the stack.

The Compare Operation

In assembly language, there is no conditional statement (if .. else ..) but instead it uses comparison operation.

The Compare Operators

Assembler	Explanation
CMP AL, BL	Set 'Z' flag if AL = BL. Set 'S' flag if AL < BL.
CMP BL, 13	Set 'Z' flag if BL = 13. Set 'S' flag if BL < 13.
CMP CL, [20]	Set 'Z' flag if CL = [20]. Set 'S' flag if CL < [20].

The Branch Operation

A branch operations are instructions that can cause application to begin executing a different instruction sequence and thus deviate from its default behavior of executing instruction in order.

The Branch Operators

Assembler		Explanation
JMP	HERE	Increase IP by 12 Decrease IP by 2 (twos complement)
JZ	THERE	Increase IP by 9 if the 'Z' flag is set. Decrease IP by 100 if the 'Z' flag is set.
JNZ	A_Place	Increase IP by 4 if the 'Z' flag is NOT set. Decrease IP by 16 if the 'Z' flag is NOT set.
JS	STOP	Increase IP by 9 if the 'S' flag is set. Decrease IP by 31 if the 'S' flag is set.
JNS	START	Increase IP by 4 if the 'S' flag is NOT set. Decrease IP by 32 if the 'S' flag is NOT set.
JO	REPEAT	Increase IP by 9 if the 'O' flag is set. Decrease IP by 33 if the 'O' flag is set.
JNO	AGAIN	Increase IP by 4 if the 'O' flag is NOT set. Decrease IP by 5 if the 'O' flag is NOT set.

The Procedures and Interrupts Operation

Procedure is a block of code that can be recalled multiple time.

Interrupts (software interrupt, hardware interrupt) is an event when an error occur and then a designed block of code is run.

ASSEMBLY LANGUAGE

The Procedure & Interrupts Operators

Assembler		Explanation
CALL	30	Save IP on the stack and jump to the procedure at address 30.
RET		Restore IP from the stack and jump to it.
INT	2	Save IP on the stack and jump to the address (interrupt vector) retrieved from RAM[02].
IRET		Restore IP from the stack and jump to it.