COAHub

Assembly Language Programming Guide Cheat Sheet

What is Assembly Language?

- Assembly language is a low-level programming language that uses mnemonic codes and labels to represent machine-level instructions.
- Each assembly instruction typically corresponds to a single machine instruction for a specific CPU architecture.

Basic Structure of an Assembly Program

```
section .data      ; Data segment (variables/constants)
section .bss       ; Uninitialized data segment
section .text      ; Code segment (instructions)
   global _start  ; Entry point for program

_start:
   ; Your instructions here
```

Common Assembly Instructions

Instruction	Description	Example
MOV	Move data	MOV AX, BX
ADD	Add values	ADD AX, 1
SUB	Subtract values	SUB AX, BX
MUL	Multiply (unsigned)	MUL BX
DIV	Divide (unsigned)	DIV BX
INC	Increment	INC AX
DEC	Decrement	DEC AX
AND	Bitwise AND	AND AX, BX

OR	Bitwise OR	OR AX, BX
XOR	Bitwise XOR	XOR AX, BX
NOT	Bitwise NOT	NOT AX
JMP	Unconditional jump	JMP label
CMP	Compare two values	CMP AX, BX
JZ/JNZ	Jump if zero/not zero	JZ label
CALL	Call procedure/subroutine	CALL proc
RET	Return from procedure	RET
PUSH	Push to stack	PUSH AX
POP	Pop from stack	POP AX

Registers (x86 Example)

Register	Purpose
AX	Accumulator
BX	Base register
CX	Counter register
DX	Data register
SI	Source index
DI	Destination index
SP	Stack pointer
BP	Base pointer
IP	Instruction pointer
FLAGS	Status flags

Addressing Modes

- **Immediate:** Operand is a constant value
  `MOV AX, 5`
- **Register:** Operand is a register
  `MOV AX, BX`
- **Direct:** Operand is a memory address
  `MOV AX, [1234h]`
- **Indirect:** Address from register
  `MOV AX, [BX]`
- **Indexed:** Uses base register plus offset
  `MOV AX, [BX+SI]`

Program Flow Control

- **Labels:** Mark positions in code
  `start:`
- **Jumps:** Change execution flow
  `JMP start`
- **Conditional Jumps:** Based on flags
  `JZ equal` (jump if zero flag set)
- **Loops:** Use `LOOP` with `CX` register

```
MOV CX, 10
```

loop_start:

; code

LOOP loop_start

```
---

**System Calls (Linux x86 Example)**

```assembly
MOV EAX, 1      ; syscall number (sys_exit)
MOV EBX, 0      ; exit code
```
```

```
INT 0x80        ; interrupt to invoke syscall
```

Procedures/Functions

```
CALL my_function
; ...
my_function:
   ; code
   RET
```

Stack Operations

- **PUSH reg:** Push register value onto stack
- **POP reg:** Pop value from stack into register

Comments

- Use ; for comments
  `MOV AX, BX ; Copy BX into AX`

Tips

- Assembly syntax and available instructions depend on the processor architecture (x86, ARM, etc.).
- Use labels for jumps and loops.
- Always keep track of the stack and registers to avoid unexpected behavior.
- Use comments liberally for clarity.

This guide provides the essential concepts and syntax for assembly language programming, suitable for quick reference and revision. For more detailed information, consult processor-specific manuals and documentation.