Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Assembly Language Programming

Dr. M Krishnam Raju
Assistant Professor
Electronics & Communication Engineering

Content

1. Fundamentals of assembly language
2. Program structure
3. Directives,
4. Programming exercises
5. Stack
6. Subroutines

Fundamentals of Assembly Language

Assembly language:

- A low-level programming language that uses human-readable instructions (mnemonics) like MOV, ADD, SUB to represent machine operations.
- Needs to be converted into machine language by an assembler.
- Easier for humans to read and write.

Machine Language:

- The most basic programming language consisting of binary code (0s and 1s) that the computer CPU can execute directly.
- Directly understood by the computer hardware; no translation needed.
- Very hard for humans to read and understand.

Fundamentals of Assembly Language

Feature	Assembly Language	Machine Language
Definition	A low-level programming language that uses human-readable instructions (mnemonics) like MOV, ADD, SUB to represent machine operations.	The most basic programming language consisting of binary code (0s and 1s) that the computer's CPU can execute directly.
Readability	Easier for humans to read and write.	Very hard for humans to read and understand.
Translation	Needs to be converted into machine language by an assembler.	Directly understood by the computer's hardware; no translation needed.
Error Detection	Easier to debug and fix errors compared to machine code.	Extremely difficult to debug.
Portability	Specific to a type of CPU (like Intel or ARM), but a little more manageable than pure machine code.	Completely tied to the hardware; not portable at all.
Examples	MOV A, B (move contents of B into A)	01010100 00100100 (binary code)

Program Structure

Initialization

- Initialize registers (clear or load them with starting values).
- Set up the Stack Pointer (SP), if the program will use the stack.
- Initialize I/O devices or memory addresses if required.

Main Program Execution

- Data transfer (moving data between memory and registers).
- Arithmetic and logical operations (ADD, SUB, AND, OR, etc.).
- Decision making (using conditional jumps like JZ, JNZ, etc.).
- Looping for repeated operations.

Subroutines

- Jump to a subroutine using CALL.
- Perform the subroutine task.
- Return to the main program using RET.

Termination

- Stop the microprocessor operation.
- Save results if needed.

Program Structure

Aspect	Initialization	Main Program Execution	Subroutines	Termination
Purpose	Set up the environment for execution	Perform the main tasks	Handle specific repetitive tasks separately	Properly stop the program
Actions	- Clear/load registers - Set Stack Pointer (SP) - Initialize I/O or memory	- Move data (register ↔ memory) - Perform arithmetic/logical operations - Decision making (branching) - Looping operations	- Use CALL to jump to subroutine - Execute subroutine logic - Return with RET to main program	- Halt microprocessor (HLT) - Optionally save final results
Instructions Used	MVI, LXI, MOV, etc.	MOV, ADD, SUB, JMP, JZ, JNZ, etc.	CALL, RET	HLT
Importance	Prepares everything; no proper run without it	Core functionality is executed here	Keeps code organized, reusable, and manageable	Prevents program from running uncontrollably
Typical Timing	Executed once at start	Executed continuously as needed	Called whenever needed inside the main program	Executed once after all tasks are complete

Directives

- In assembly language, directives (also called pseudo-operations or pseudo-ops) are instructions for the assembler, not for the microprocessor.
- They tell the assembler how to process the code, organize memory, or handle variables — but they do not translate into machine code that the CPU executes.

ORG (Origin)

- Sets the starting address for the program or data.
- Syntax: ORG 2000H
- Example: If you write ORG 2000H, the next code/data will be placed starting from memory address 2000H.

END

- Marks the end of the source code.
- After this, the assembler ignores anything written.
- Syntax: END
- Example: Typically the last line in an assembly program.

Programming exercises

Write an assembly language program to add two 8 bit numbers stored at address 2050 and address 2051 in 8085 microprocessor. The starting address of the program is taken as 2000.

- Load the first number from memory location 2050 to accumulator.
- Move the content of accumulator to register H.
- Load the second number from memory location 2051 to accumulator.
- Then add the content of register H and accumulator using “ADD” instruction and storing result at 3050
- The carry generated is recovered using “ADC” command and is stored at memory location 3051

Programming exercises

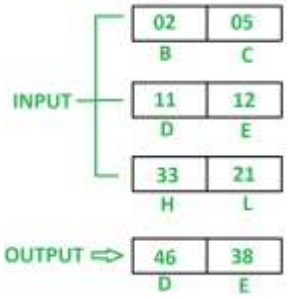
Memory Address	Mnemonics	Comment
2000	LDA 2050	A<-[2050]
2003	MOV H, A	H<-A
2004	LDA 2051	A<-[2051]
2007	ADD H	A<-A+H
2008	MOV L, A	L←A
2009	MVI A 00	A←00
200B	ADC A	A←A+A+carry
200C	MOV H, A	H←A
200D	SHLD 3050	H→3051, L→3050
2010	HLT	

Programming exercises

- LDA 2050 moves the contents of 2050 memory location to the accumulator.(Opcode: 3A 50 20)
- MOV H, A copies contents of Accumulator to register H to A(Opcode: 67)
- LDA 2051 moves the contents of 2051 memory location to the accumulator.(Opcode: 3A 51 20)
- ADD H adds contents of A (Accumulator) and H register (F9). The result is stored in A itself. For all arithmetic instructions A is by default an operand and A stores the result as well(Opcode: 84)
- MOV L, A copies contents of A (34) to L(Opcode: 6F)
- MVI A 00 moves immediate data (i.e., 00) to A(Opcode: 3E 00)
- ADC A adds contents of A(00), contents of register specified (i.e A) and carry (1). As ADC is also an arithmetic operation, A is by default an operand and A stores the result as well(Opcode: 8F)
- MOV H, A copies contents of A (01) to H (Opcode: 67)
- SHLD 3050 moves the contents of L register (34) in 3050 memory location and contents of H register (01) in 3051 memory location(Opcode: 22 50 30)
- HLT stops executing the program and halts any further execution(Opcode: 76)

Programming exercises

Write an assembly language program to add three 16 bit numbers stored in register HL, DE, BC and store the result in DE with minimum number of instructions.



Assumptions:

- Numbers to be added are already stored in register HL, DE, BC
- Numbers stored in register are such that final result should not be greater than FFFF

DAD instruction take one argument and that argument can be register B, D, H or SP XCHG instruction exchanges the content of register D with H and E with L Algorithm

- Add the content of DE register in HL and store the result in HL by help of DAD instruction
- Move the content of register B in D and C in E
- Repeat step 1
- Use XCHG instruction to swap the content of DE with HL. We will get the result in DE

Programming exercises

Memory Address	Mnemonics	Comment
2000	DAD D	H <- H + D, L <- L + E
2001	MOV D, B	D <- B
2002	MOV E, C	E <- C
2003	DAD D	H <- H + D, L <- L + E
2004	XCHG	Swap content of HL with DE
2005	HLT	END

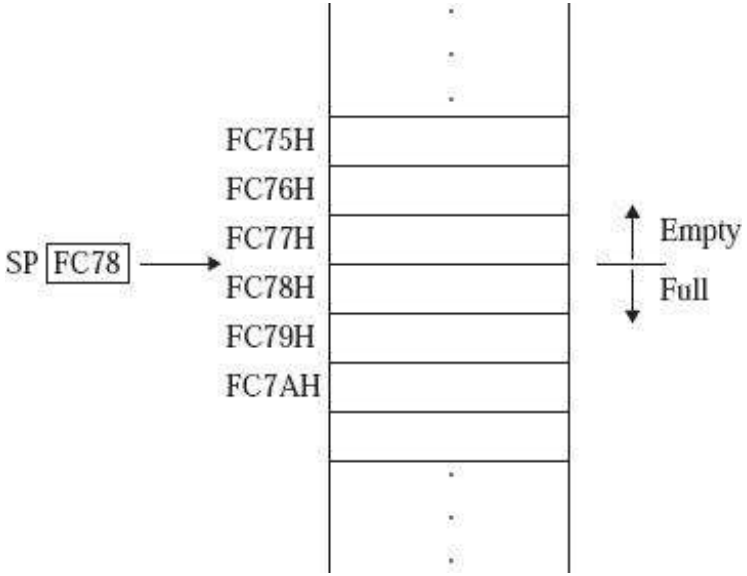
- DAD D – adds the content of register D in H and register E in L and store the result in HL
- MOV D, B – moves the value of register B in register D
- MOV E, C moves the value of register C in register E
- Same as step 1
- XCHG – exchange the content of register H with register D and L with E.
- HLT – stops executing the program and halts any further execution

Stack and Subroutine

- Stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine.
- Then the return address used to get pushed on this stack. Also to swap values of two registers and register pairs we use the stack as well.
- Stack Pointer register will hold the address of the top location of the stack.
- SP is a special purpose 16-bit register. It contains a memory address.
- Suppose SP contents are FC78H, then the 8085 interprets it as follows. Memory locations FC78H, FC79H, ..., FFFFH are having useful information. These locations are treated as filled locations. Memory locations FC77H, FC76H, ..., 0000H are not having any useful information.
- On a stack, we can perform two operations. PUSH and POP. In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack. On the other hand, in case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2. The contents of SP specify the top most useful location in the stack.

Stack and Subroutine

Instruction	Purpose
PUSH rp	Save register pair onto the stack
POP rp	Retrieve register pair from the stack
CALL addr	Push the address of the next instruction onto the stack and jump
RET	Pop address from stack and jump back
XTHL	Exchange top of stack with H and L registers
SPHL	Copy HL register content to SP



Example

LXI SP, 5000H; Set stack pointer to 5000H
MVI A, 32H; Move 32H into accumulator A
PUSH PSW; Push A and flags onto stack (PSW = Accumulator + Flag register)
POP B; Pop two bytes from stack into register pair BC

Stack and Subroutine

- A subroutine in the 8085 microprocessor is a separate block of code designed to perform a specific task and can be invoked (called) from the main program at one or multiple points. Instead of duplicating code, the main program branches to the subroutine, executes its instructions, and then returns to the instruction immediately following the call.
- **CALL Instruction:** The 8085 uses the CALL instruction to invoke a subroutine. CALL saves the address of the next instruction (return address) onto the stack and transfers program control to the specified memory address of the subroutine.
- **RET Instruction:** The RET instruction is used at the end of the subroutine to retrieve the return address from the stack and continue program execution from that point.
- **Nested Subroutines:** The 8085 allows subroutines to call other subroutines, known as nesting. The stack keeps track of multiple return addresses in such cases.
- Data can be passed to a subroutine via registers, memory locations, or the stack.

Stack and Subroutine

Example:

```
MVI H, 30H          Load H register with 30
MVI L, 40H          Load L register with 40
CALL ADDITION       Call the ADDITION subroutine

Main Program continues here
ADDITION:           Subroutine for addition
MOV A, H            Move value in H register to accumulator
ADD L               Add value in L register to accumulator
RET                 Return from subroutine
```

In the example above:

- The CALL ADDITION instruction causes the program to jump to the subroutine labeled ADDITION.
- After the addition is performed, the RET instruction brings control back to the main program.

Parul® University

NAAC GRADE A++

https://paruluniversity.ac.in/