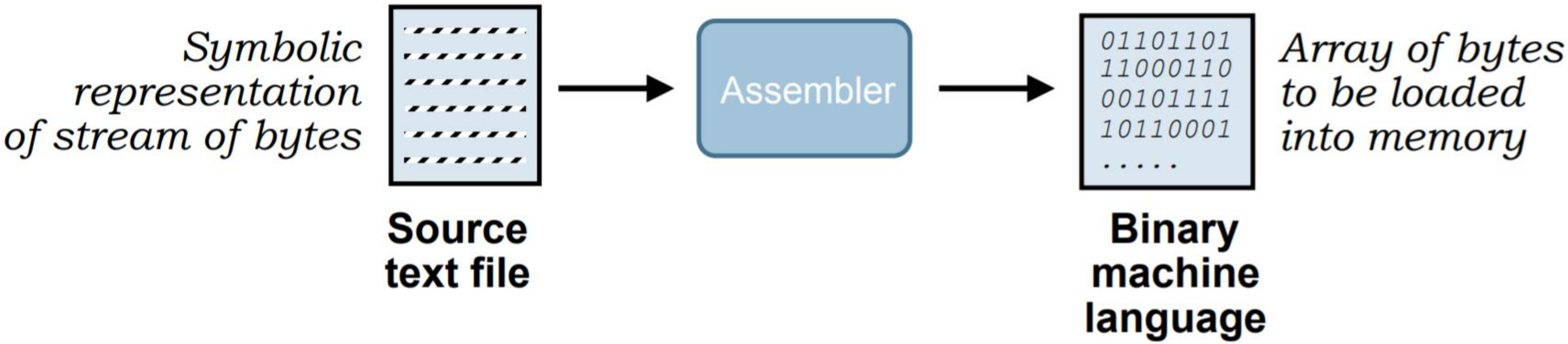
Assembler Design

Nahin Ul Sadad
Lecturer
CSE, RUET

Assembly Language

Assembly language abstracts bit-level representation of instructions and addresses. Assembler converts assembly language to machine language.



Different phases of Assembler

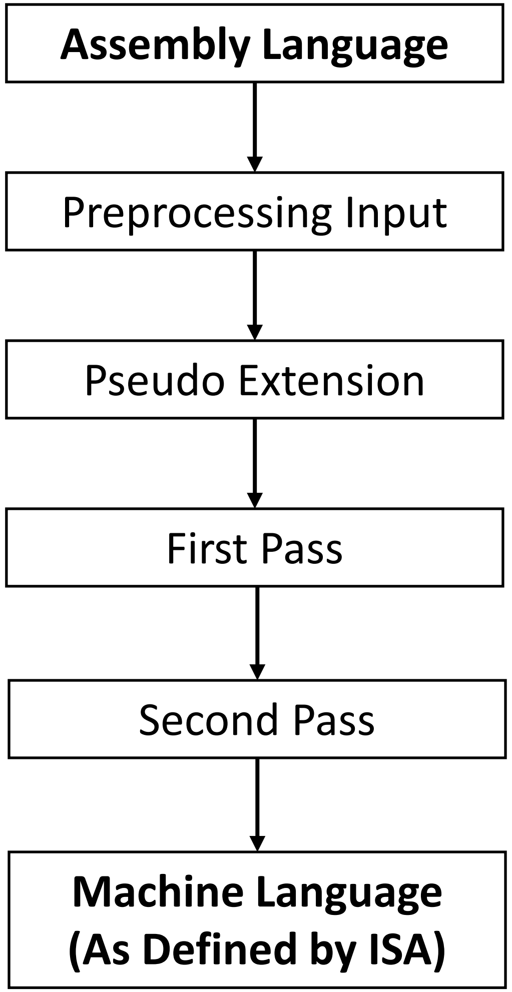


Figure: Different phases of Assembler

Different phases of Assembler

1. Preprocessing Input: It will remove all the unnecessary contents of program and tokenize instructions. It performs following tasks:

- Remove comment.
- Replace all consecutive white-spaces into one white-space.
- Remove all white-spaces before and after instruction.
- Tokenize every word in instruction.

2. Pseudo Extension: All pseudo-instructions in assembly program will be extended. For example: **`MOV R1, R2`** will be replaced by **`XOR R1, R1`** & **`ADD R1, R2`**.

Different phases of Assembler

3. Pass 1: All assembly instructions will be converted to machine instructions except instruction containing labels. It will also construct `machine code` table and `label` tables. It performs following tasks:

- Translate every instruction to its corresponding machine code except jump instruction.
- Find all the label outside instruction and store them in a label table.
- Store all instructions in a `machine code` table which contains opcode type, operands, its address and equivalent machine code.

4. Pass 2: Whole assembly program will be translated to machine code with appropriate addressing. It performs following tasks:

- Translate every jump instruction to its corresponding machine code.
- Show error message if label’s address is not found in symbol table.

Example 1: Sample Code with registers

Question: Write assembly code from following pseudo code and convert that to machine code.

Pseudo Code
`R4 = 5` `R5 = 6` `IF R4 >= R5 THEN` `      R4 = R4 + 1` `ELSE` `      R5 = R5 + 1` `FINISH (NO OPERATION LEFT)`

Here, R4 and R5 are registers.

Example 1: Sample Code with Registers

Answer:

Translating pseudo code to assembly code:

Code

```
      XOR R4, R4   #clearing R4
      XOR R3, R3   #clearing R3

      ADD R4, 5    #R4 = 5
      ADD R3, 6    #R3 = 6

      CMP R4, R3   #comparing R4 and R3 and it will set zf and sf flag
      JGE IF_      #since 5<6, zf=0 and sf=1 and Condition is false.
      ADD R3, 1    #R3 = R3 + 1
      JMP ELSE_EXIT:     #JMP to exit to skip instructions of IF part
IF_:
      ADD R4, 1    #R4 = R4 + 1
ELSE_EXIT:
      JMP ELSE_EXIT      #NO OPERATION (NOP).
```

Perform different phases of assembler into this code.

Example 1: Sample Code with Registers

Step 1: Preprocessing inputs

Address (Logical Address)	Code
00	`XOR R4, R4`
01	`XOR R3, R3`
02	`ADD R4, 5`
03	`ADD R3, 6`
04	`CMP R4, R3`
05	`JGE IF_`
06	`ADD R3, 1`
07	`JMP ELSE_EXIT`
08	`IF_: ADD R4, 1`
09	`ELSE_EXIT: JMP ELSE_EXIT`

Example 1: Sample Code with Registers

Step 2: Pass 1 (Ignoring Pseudo Extension)

Address (Logical Address)	Code
00	0000101001000
01	0000100110110
02	0101011000101
03	0101010110110
04	0010111000110
05	100110**XXXXXXX**
06	0101010110001
07	100000**XXXXXXX**
08	0101011000001
09	100000**XXXXXXX**

Label Table

LABEL	Address
IF_	08 (0001000)
ELSE_EXIT	09 (0001001)

Example 1: Sample Code with Registers

Step 3: Pass 2

Address (Logical Address)	Code
00	0000101001000
01	0000100110110
02	0101011000101
03	0101010110110
04	0010111000110
05	100110**<u>0001000</u>**
06	0101010110001
07	100000**<u>0001001</u>**
08	0101011000001
09	100000**<u>0001001</u>**

Example 1: Sample Code with Registers (Conversion Details) [No need to add this in exam]

Address (Logical Address)	Code	Opcode (2 bits)	Opcode (4 bits)	Register 1	Register 2	Constant	Address	Machine Code												
								12	11	10	9	8	7	6	5	4	3	2	1	0
0000000 (0)	XOR R4, R4	00 (A&L Reg)	0010 (XOR)	100 (R4)	100 (R4)	X	X	0	0	0	0	1	0	1	0	0	1	0	0	0
0000001 (1)	XOR R3, R3	00 (A&L Reg)	0010 (XOR)	011 (R3)	011 (R3)	X	X	0	0	0	0	1	0	0	1	1	0	1	1	0
0000010 (2)	ADD R4, 5	01 (A&L Imm)	0101 (ADD)	100 (R4)	X	0101 (5)	X	0	1	0	1	0	1	1	0	0	0	1	0	1
0000011 (3)	ADD R3, 6	01 (A&L Imm)	0101 (ADD)	011 (R3)	X	0110 (6)	X	0	1	0	1	0	1	0	1	1	0	1	1	0
0000100 (4)	CMP R4, R3	00 (A&L Reg)	1011 (CMP)	100 (R4)	011 (R3)	X	X	0	0	1	0	1	1	1	0	0	0	1	1	0
0000101 (5)	JGE IF_	10 (Branching)	0110 (JGE)	X	X	X	0001000 (8)	1	0	0	1	1	0	0	0	0	1	0	0	0
0000110 (6)	ADD R3, 1	01 (A&L Imm)	0101 (ADD)	011 (R3)	X	0001 (1)	X	0	1	0	1	0	1	0	1	1	0	0	0	1
0000111 (7)	JMP ELSE_EXIT	10 (Branching)	0000 (JMP)	X	X	X	0001001 (9)	1	0	0	0	0	0	0	0	0	1	0	0	1
0001000 (8)	IF_: ADD R4, 1	01 (A&L Imm)	0101 (ADD)	100 (R4)	X	0001 (1)	X	0	1	0	1	0	1	1	0	0	0	0	0	1
0001001 (9)	ELSE_EXIT: JMP ELSE_EXIT	10 (Branchin)	0000 (JMP)	X	X	X	0001001 (9)	1	0	0	0	0	0	0	0	0	1	0	0	1

Example 2: Sample Code with Variables/Memory locations

Question: Write assembly code from following pseudo code and convert that to machine code.

Pseudo Code
`A = 5` `B = 6` `IF A < B THEN` `      A = A + 2` `FINISH (NO OPERATION LEFT)`

(Since nothing is mentioned about A and B, A and B are memory locations/variables.)

Example 2: Sample Code with Variables/Memory locations

Answer:

Translating pseudo code to assembly code:

Code

```
        XOR R4, R4        #clearing R4
        XOR R3, R3        #clearing R3

        ADD R4, 5         #R4 = 5
        ADD R3, 6         #R3 = 6

        STORE [A], R4     #Storing contents of R4 in Memory location A in RAM
        STORE [B], R3     #Storing contents of R3 in Memory location B in RAM

        XOR R0, R0        #clearing R0
        XOR R1, R1        #clearing R1

        LOAD R0, [A]      #Loading contents of Memory location A (RAM) in R0
        LOAD R1, [B]      #Loading contents of Memory location B (RAM) in R1

        CMP R0, R1        #comparing R0 and R1 and it will set zf and sf flag
        JL IF_            #since 5<6, sf=0 and Condition is true.
        JMP ELSE_EXIT     #JMP to exit to skip instructions of IF part
IF_:
        ADD R0, 2         #R0 = R0 + 2
        STORE [A], R0
ELSE_EXIT:
        JMP ELSE_EXIT     #NO OPERATION (NOP).
```

Example 2: Sample Code with Variables/Memory locations

Translating pseudo code to assembly code:

Code

```
        XOR R4, R4        #clearing R4
        XOR R3, R3        #clearing R3

        ADD R4, 5         #R4 = 5
        ADD R3, 6         #R3 = 6

        STORE [0], R4     #Storing contents of R4 in Memory location 0 in RAM
        STORE [1], R3     #Storing contents of R3 in Memory location 1 in RAM

        XOR R0, R0        #clearing R0
        XOR R1, R1        #clearing R1

        LOAD R0, [0]      #Loading contents of Memory location 0 (RAM) in R0
        LOAD R1, [1]      #Loading contents of Memory location 1 (RAM) in R1

        CMP R0, R1        #comparing R0 and R1 and it will set zf and sf flag
        JL IF_            #since 5<6, sf=0 and Condition is true.
        JMP ELSE_EXIT     #JMP to exit to skip instructions of IF part
IF_:
        ADD R0, 2         #R0 = R0 + 2
        STORE [0], R0
ELSE_EXIT:
        JMP ELSE_EXIT     #NO OPERATION (NOP).
```

Perform different phases of assembler into this code.

Example 2: Sample Code with Variables/Memory locations

Step 1: Preprocessing inputs

Address (Logical Address)	Code
00	`XOR R4, R4`
01	`XOR R3, R3`
02	`ADD R4, 5`
03	`ADD R3, 6`
04	`STORE [0], R4`
05	`STORE [1], R3`
06	`XOR R0, R0`
07	`XOR R1, R1`
08	`LOAD R0, [0]`
09	`LOAD R1, [1]`
10	`CMP R0, R1`
11	`JL IF_`
12	`JMP ELSE_EXIT`
13	`IF_: ADD R0, 2`
14	`STORE [0], R0`
15	`ELSE_EXIT: JMP ELSE_EXIT`

Example 2: Sample Code with Variables/Memory locations

Step 2: Pass 1 (Ignoring Pseudo Extension)

Address (Logical Address)	Code
00	0000101001000
01	0000100110110
02	0101011000101
03	0101010110110
04	1100111000000
05	1100110110001
06	0000100000000
07	0000100010010
08	1100000000000
09	1100000010001
10	0010110000010
11	100011**xxxxxxx**
12	100000**xxxxxxx**
13	0101010000010
14	1100110000000
15	100000**xxxxxxx**

Label Table

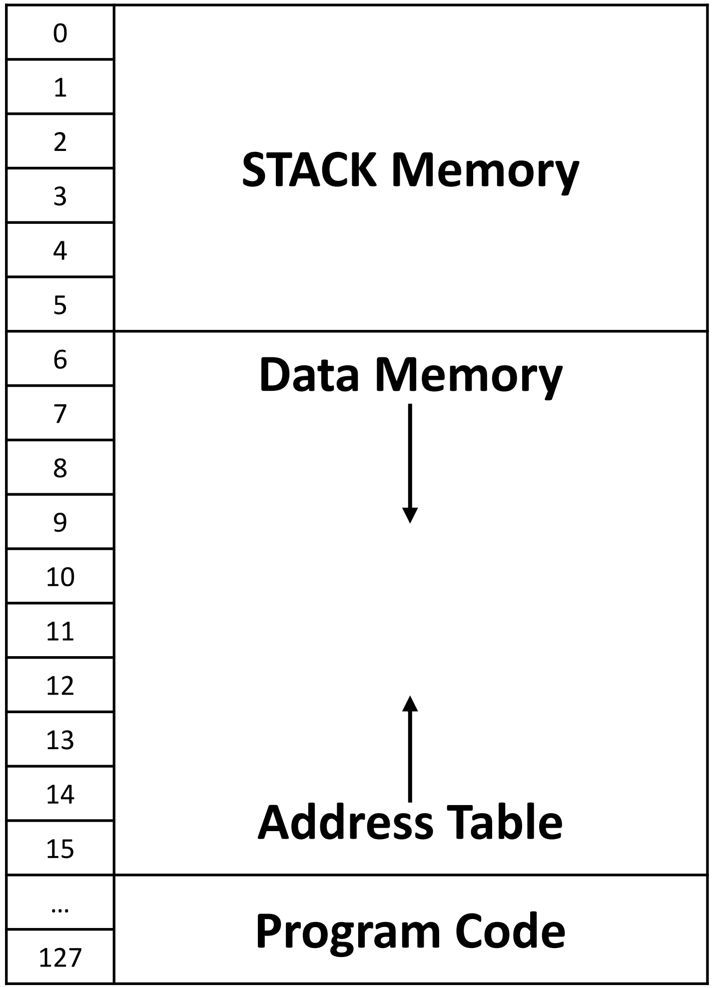
LABEL	Address
IF_	13 (0001101)
ELSE_EXIT	15 (0001111)

Example 2: Sample Code with Variables/Memory locations

Step 2: Pass 2

Address (Logical Address)	Code
00	0000101001000
01	0000100110110
02	0101011000101
03	0101010110110
04	1100111000000
05	1100110110001
06	0000100000000
07	0000100010010
08	1100000000000
09	1100000010001
10	0010110000010
11	100011**0001101**
12	100000**0001111**
13	0101010000010
14	1100110000000
15	100000**0001111**

Simple Process Model in CPU



1. **Stack Memory:** This STACK memory is used for passing parameters to functions/procedures. (Address range 0-5)
2. **Data Memory:** Data memory is used to store local variables. It grows downward. (Address range 6-15)
3. **Address Table:** Address table is used to store addresses. It grows upward. (Address range 15-6)
4. **Program Code:** Program Code is the program itself. (Address range 16-127)

STACK Memory

In order to send arguments to the called function, arguments are saved into STACK memory. Return values are also saved into STACK memory. Data inside Local variables/registers are also saved into STACK before calling function because that functions will also use local variables and registers.

STACK memory is called STACK memory because it works like a STACK/Last-In-First-Out (LIFO). It uses PUSH and POP operations to store and retrieve arguments. **STACK Pointer (SP) register** keep track of location of last data inside STACK.

1. **PUSH (X):** Push X value to STACK memory.
2. **POP:** Pop/Retrieve last stored data from STACK memory.

Sample Code: Using STACK Memory in Procedures

Pseudo Code

```
main()
{
   D = 5                #Local Variables
   E = add(1,2)         #Two arguments 1(A)&2(B)
   D = D+1
}

add(A,B)
{
  C = A + B
  return C
}
```

Sample Code: Using STACK Memory in Procedures

Pseudo Code (STACK Memory)	
``main() { D = 5 #Local Variables PUSH(D) PUSH(2) #B PUSH(1) #A PUSH(RETURN_ADDRESS) CALL add POP(RETURN_VALUE) E = RETURN_VALUE POP(D) D = D+1 }```	```add(A,B) { POP(RETURN_ADDRESS) POP(A) #1 POP(B) #2 C = A + B PUSH(C) #C RETURN RETURN_ADDRESS }```

Interrupts

An interrupt is a response by the processor to an event that needs attention from the software.

An interrupt condition alerts the processor and serves as a request for the processor to interrupt the currently executing code when permitted, so that the event can be processed in a timely manner.

If the request is accepted, the processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event.

This interruption is temporary, and, unless the interrupt indicates a fatal error, the processor resumes normal activities after the interrupt handler finishes.

There are two types of Interrupts:

1. Software Interrupts
2. Hardware Interrupts

Hardware Interrupts

Hardware Interrupt is caused by some hardware device such as request to start an I/O, a hardware failure or something similar. Hardware interrupts were introduced as a way to avoid wasting the processor's valuable time in polling loops, waiting for external events.

For example, when an I/O operation is completed such as reading some data into the computer from a tape drive.

Software Interrupts

Software Interrupt is invoked by the use of INT instruction. This event immediately stops execution of the program and passes execution over to the INT handler. The INT handler is usually a part of the operating system and determines the action to be taken. It occurs when an application program terminates or requests certain services from the operating system.

For example, output to the screen, execute file etc.

Exercises

1. Convert following assembly code to machine code:

```
          XOR R0, R0
          XOR R1, R1

          ADD R0, 5
          ADD R1, 6

          STORE [C], R4
          STORE [D], R3

          XOR R3, R3
          XOR R4, R4

          LOAD R3, [C]
          LOAD R4, [D]

          CMP R3, R4
          JGE LABEL1
          JMP LABEL2
LABEL1:
          ADD R3, 2
          STORE [A], R0
LABEL2:
          JMP LABEL1
```

Exercises

2. Write assembly code from following pseudo code:

Pseudo Code
`NUM1 = 1` `NUM2 = 2` `IF A > B THEN` `        NUM1 = NUM2 + 2` `ELSE` `        NUM2 = NUM1 - 1` `FINISH (NO OPERATION LEFT)`

3. Write assembly code from following pseudo code and convert it to machine code:

Pseudo Code
`R0 = 5` `R2 = 6` `IF R0 < R2 THEN` `        R4 = R0 AND R2` `FINISH (NO OPERATION LEFT)`

Here, **R4** and **R5** are registers.

Thank you ☺