Activity No. 2

C PROGRAM TRANSLATION TO ASSEMBLY LANGUAGE

1. Objective:

To translate a C program into its equivalent Assembly program

2. Intended Learning Outcomes (ILOs):

The students should be able to:

2.1 Relate a C program to assembly program

2.2 Translate a C program to assembly program

3. Discussion :

INTRODUCTION TO ASSEMBLY LANGUAGE

Assembly language is the basis of the C programming language that is why a program in C can be easily translated in assembly language.

The C programming language follows a certain program structure. Figure 2.1 shows a C program structure. The program in Figure 2.1 absolutely does nothing but it a complete executable file and shows a typical program in C.

```
#include<stdio.h>
#include<conio.h>
main()
{
// Local variable declaration;
return 0;
}
```

Figure 2.1-The Program structure of C program

Just like programming in C, it is good programming practice to develop some sort of standard by which we write our programs in assembly. Figure 2.2 and Figure 2.3 show the program structure of .COM and .EXE respectively.

```
MAIN SEGMENT
ASSUME
SS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h
START:
INT 20
MAIN ENDS
END START
```

Figure 2.2- The Program Structure of .COM program

MAIN SEGMENT - declares a segment called MAIN. A .COM file must fit into 1 segment of memory (usually 65535 bytes)
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN - tells the assembler the initial values of the CS,DS,ES, and SS register. Since a .COM must fit into one segment they all point to the segment defined in the line above.
ORG 100h - Since all .COM files start at CS:0100 you declare the entry point to be 100h.
START: -a label.
INT 20 - Returns to DOS
MAIN ENDS - Ends the MAIN segment
END START - Ends the START label

```
DOSSEG
.MODEL SMALL
.STACK 200h
.DATA
.CODE
START:
INT 20
END START
```

Figure 2.3- The Program Structure of .EXE program

DOSSEG - sorts the segments in the order: Code,Data,Stack
.MODEL SMALL - selects the SMALL memory model, available models are:

TINY: All code and data combined into one single group called DGROUP. Used for .COM files.
SMALL: Code is in a single segment. All data is combined in DGROUP. Code and data are both smaller than 64k. This is the standard for standalone assembly programs.
MEDIUM: Code uses multiple segments, one per module. All data is combined in DGROUP. Code can be larger than 64k, but data has to be smaller than 64k
COMPACT: Code is in a single segment. All near data is in DGROUP. Data can be more than 64k, but code cannot.
LARGE: Code uses multiple segments. All near data is in DGROUP. Data and code can be more than 64k, but arrays cannot.
HUGE: Code uses multiple segments. All near data is in DGROUP. Data, code and arrays can be more than 64k. Most of the time you want to use SMALL to keep the code efficient.
.**STACK 200h** - sets up the stack size. In this case 200h bytes
.**DATA** - The data segment. This is where all your data goes (variables for example).
.**CODE** - The code segment. This is where your actually program goes.
START: - Just a label.

INT 20 - exits the program.
END START – ends the start label.

Conditional Statements

Conditional statements are programming language statement that selects an execution path based on whether a condition is TRUE or FALSE. When applied in assembly there are three things that we must consider, these are:

Label

Acts as an identifier that acts as a place marker for instructions and data. When placed just before an instruction implies the instruction's address. If placed just before a variable implies the variable's address. START: is a label.

Compare command and Jump instructions

The compare (cmp) jump instructions are the instructions that are used for conditional statements, they are listed in Table 2.1

CMP	Compare
JA	Jump if Above
JAE	Jump if Above or Equal
JB	Jump if Below
JBE	Jump if Below or Equal
JC	Jump on Carry
JCXZ	Jump if CX is Zero
JE	Jump if Equal
JG	Jump if Greater
JGE	Jump if Greater than or Equal
JL	Jump if Less than
JLE	Jump if Less than or Equal
JMP	Jump unconditionally
JNA	Jump if Not Above

JNAE	Jump if Not Above or Equal
JNB	Jump if Not Below
JNE	Jump if Not Equal
JNG	Jump if Not Greater
JNGE	Jump if Not Greater or Equal
JNL	Jump if Not Less
JNLE	Jump if Not Less or Equal
JNO	Jump if No Overflow
JNP	Jump on No Parity
JNS	Jump on No Sign
JNZ	Jump if Not Zero
JO	Jump on Overflow
JP	Jump on Parity
JPE	Jump on Parity Even
JPO	Jump on Parity Odd
JS	Jump on Sign
JZ	Jump on Zero

Table 2.1- The Compare and Jump Instructions

Loops

Loops or repetition allow a set of instructions to be repeated until certain condition is reached. The two most common loop types are the FOR and WHILE statements.

Variables

Variables are simply a name given to a memory area that contains data. To access that data you do not have to specify that memory address, you can simply refer to that variable. Variables can consist of number, characters and underscores (_), but must begin with a character.

The three most common variables types are the BYTES, WORD, DOUBLE WORD.

To declare variables follow the format:
NAME TYPE CONTENTS

where: TYPE-is either DB (Declare Byte),
DW (Declare Word), or
DD (Declare DoubleWord).

Constants

Constants are data types that like variables can be used in your program to access data stored at specific memory locations, but unlike variables they cannot be changed during the execution of a program.

To declare constants use EQU,

Example 2.1
```
constant EQU DEADh
```

Example 2.1 declares a constant called constant and sets it equal to 100, then it assigns the value in constant to dx and ax and adds them.

This is the same as,

```
mov dx,100
mov ax,100
add dx,ax
```

Arrays

Example 2.2 shows how to create a 5 byte long array called my_String and sets it equal to the string “I love Assembly Programming!”

Example 2.2
```
my_String DB " I love Assembly Programming!$"
```

The symbol “ $” must be present at the end, otherwise computer will start executing 11instructions after the last character, which is whatever is in memory at that particular location.

Single quote or double quote can be used within a string like in Example 2.3

Example 2.3
```
Cow DB 'John said " I love Assembly Programming!"$'
or
Cow DB "John said I love Assembly Programming!!'$"
```

Use whichever you think looks better. What if you have to use both types of quotes? Example 2.4 will show you how.

Example 2.4
Cow DD 'John said "I say: ""'GNAAAARF!""$'

Use double double/single quotes.

What if you do not know what the variable is going to equal? Then you may use the format shown in Example 2.5.

Example 2.5
Uninitialized_variable DB ?

ASSEMBLER
For us to be able to create programs in assembly an assembler is needed. The process for the creation of an assembly program passes through the following phases:
WRITING. Through an EDITOR (i.e. Notepad), the source code with extension .ASM.
ASSEMBLING. Through an ASSEMBLER(TASM-Turbo macro Assembler), of the single source files and generation of as many object files, with extension .OBJ.
CREATION OF THE EXECUTABLE FILE. Through a LINKER(TLINK), with extension .EXE.
CHECK and CORRECTION of ERRORS, if any. Through a DEBUGGER.

4. Resources:

PC
TASM
C compiler (optional)

5. Procedure:

1. Open Notepad or any text editor, and write the following:

```
 MAIN SEGMENT
 ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h
START:
        MOV DL,41h
        MOV DH,41h
        CMP DH,DL
        JE TheyAreEqual
        JMP TheyAreNotEqual
TheyAreNotEqual:
    MOV AH,02h
    MOV DL,4Eh
    INT 21h
    INT 20h
TheyAreEqual:
```

```
        MOV AH,02h
        MOV DL,59h
        INT 21h
        INT 20h
MAIN ENDS
END START
```

2. Save the program as Prog2_1.asm.
3. Open TASM at the command prompt, type the following prompt in your computer:

   C:\>cd Tasm[Enter]
4. Assemble the file using TASM. At the c:\Tasm>, type
   C:\Tasm>**Tasm Prog2_1.asm**[**Enter]**
5. If the file assembled successfully, link the file by typing,
   C:\Tasm>**Tlink/t Prog2_1.obj**[**Enter]**
6. Run the program,
   C:\Tasm>**Prog2_1**[**Enter]**
7. Record your output in the Data and Results provided.
8. Prog2_1 is equivalent to the following C code.

```
#include<stdio.h>
#include<conio.h>
main()
{
int DH,DL;
DL = 41;
DH = 41;
if (DH == DL)
printf("Y");
else
printf("N");
getch();
return 0;
}
```

9. Examine each of the following C codes and assembly codes if they produce the same output.

;Prog2_2.asm	//Prog2_2.c

MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
	#include<stdio.h>
	#include<conio.h>
	main()

	ORG 100h
	start:
	MOV CX,5 ;set CX equal to 5
	LOOP_LABEL:
	MOV AH,02h ;writes on screen
	MOV DL,2Ah ;prints the character
	INT 21h
	LOOP LOOP_LABEL
	INT 20h
	MAIN ENDS
	END START
		{
		int cx;
		for (cx=0;cx<5; cx++)
		printf("*");
		getch();
		return 0;
		}

	;Prog2_3.asm
	
	MAIN SEGMENT
	ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
	ORG 100h
	start:
	MOV CX, 5
	loop_label:
	MOV AH,02h
	MOV DL,2Ah
	INT 21h
		//Prog2_3.c
		
		#include<stdio.h>
		#include<conio.h>
		main()
		{
		void print();
		print();
		getch();
		return 0;
		}

DEC CX ;decrement CX
CMP CX,0 ;check if CX is zero
JNZ loop_label ;
INT 20h
MAIN ENDS
END START
	void print()
	{
	int cx=1;
	while (cx<=5){
	printf("*");
	cx++;}
	}

;Prog2_4.asm
 	
MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h
START:
message DB "Hello World!$"
MOV AH,09h
MOV DX,OFFSET message
INT 21h
INT 20h
MAIN ENDS
END START
	//Prog2_4.c
	
	#include<stdio.h>
	#include<conio.h>
	main()
	{
	char message[]="Hello World!";
	printf("%s",message);
	getch();
	return 0;
	}

10. Record your output in the Data and Result provided.

Course: BET-CpET	**Activity No.: 2**
Group No.:	**Section: 2A**
Group Members:	**Date Performed: 03/02/2025**
	Date Submitted:03/04/2025
	Instructor: Ma'am De los Trinos

6. DATA AND RESULTS:

1. For the program:

```
MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h
START:
        MOV DL,41h
        MOV DH,41h
        CMP DH,DL
        JE TheyAreEqual
        JMP TheyAreNotEqual
TheyAreNotEqual:
        MOV AH,02h
        MOV DL,4Eh
        INT 21h
        INT 20h
TheyAreEqual:
        MOV AH,02h
        MOV DL,59h
        INT 21h
        INT 20h
MAIN ENDS
END START
```

The following result is:

```
YC:\>
```

Where in Y is the output of the program since DL and DH are equal to each other hence the JE function is executed which contains commands in the TheyAreEqual Label. Because there is no new line function commands in the TheyAreEqual function, the output "Y" is near the current file directory of the programmer's DOSBOX. Furthermore, the C equivalent code also yields the expected result of "Y."

2. The following Assembly Program:

```
MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h
```

```
START:
    MOV CX, 5
LOOP_LABEL:
    MOV AH,02h
    MOV DL,2Ah
    INT 21h
    LOOP LOOP_LABEL
    INT 20h

MAIN ENDS
END START
```

and C Program:

```
#include <stdio.h>
#include <conio.h>
int main(){
    int cx;
    for (cx=0;cx<5;cx++)
    printf("*");
    getch();

    return 0;
}
```

Loops the printed character "*****" when executed. Due to the fact that the assembly code provided did not have any functions for adding a newline, the output of the .com program was printed beside the current file directory of the DOSBOX Assembler. The output results as follows: *****C:\>

3. The following Assembly Code:

```
MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN
ORG 100h

START:
    MOV CX, 5

LOOP_LABEL:
    MOV AH,02h
    MOV DL,2Ah
    INT 21h
    DEC CX
    CMP CX, 0
    JNZ loop_label;
```

```
    INT 20h

MAIN ENDS
END START
```

and C program:

```
#include <stdio.h>
#include <conio.h>
int main(){
    void print();
    print();
    getch();
    return 0;
}

void print(){
    int cx=1;
    while (cx<=5){
        printf("*");
        cx++;
    }
}
```

Both print "*****". Due to the fact that the assembly code does not include functions for new line after printing the output, the result presents the following. *****C:\>

4. The Assembly Code:

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
    message DB "Hello World!$"
    MOV AH,09h
    MOV DX,OFFSET message
    INT 21h
    INT 20h
MAIN ENDS
END START
```

and the C Program:

```
#include <stdio.h>
#include <conio.h>
```

```
int main(){
    char message[]= "Hello World!";
    printf("%s", message);
    getch();
    return 0;
}
```

Both print the string Hello World!. Due to the provided assembly program not having a new line function added, the follow shows the output when ran in DOSBOX: Hello World!C:\>

Problem 1:

```
#include <stdio.h>
int main(){
    int i=0;
    for (i=0;i<=6;i++){
        printf("Assembly Programming is EASY!");
    }
    return 0;
}
```

Problem 2:
Letter a.

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
    MOV AX, 2
    MOV BX, 3
    CMP AX, BX
    JBE LessorEqual

LessorEqual:
    MOV AX, 5
    MOV BX, 6

    MOV AH, 09h
    LEA DX, MESSAGE
    INT 21H

    INT 20H

MESSAGE DB "EAX= 5, EBX=6", 0DH, 0AH, "$"
```

```
MAIN ENDS
END START
```

Letter b.

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
    MOV DH, VAR1
    MOV DL, VAR2
    CMP DH, DL
    JBE LessorEqual
    JMP Greater

LessorEqual:
    MOV DL, VAR3

    MOV AH, 09H
    LEA DX, MESSAGE
    INT 21H

    INT 20H
Greater:
    MOV BYTE PTR VAR3, 10
    MOV DL, VAR4

    MOV AH, 09H
    LEA DX, MESSAGE2
    INT 21H

    INT 20H

VAR1 DB 2
VAR2 DB 3
VAR3 DB 15
VAR4 DB 20

MESSAGE DB "VAR3= 15", 0DH, 0AH, "$"
MESSAGE2 DB "VAR3= 10, VAR4= 20", 0DH, 0AH, "$"

MAIN ENDS
END START
```

Letter c.

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
   MOV AL, 5
   MOV BL, 4
   CMP AL, BL
   JBE Error

   MOV CL, 4
   CMP BL, CL
   JNE Error

   JMP Success

Success:
   MOV DL, x
   MOV AH, 09H
   LEA DX, SUCCESSMESSAGE
   INT 21H

   INT 20H

Error:
   MOV AH, 09H
   LEA DX, FAILMESSAGE
   INT 21H

   INT 20H

SUCCESSMESSAGE DB "x= 1", 0DH, 0AH, "$"
FAILMESSAGE DB "ERROR: A CONDITION IS NOT MET!", 0DH, 0AH, "$"
x DB 1

MAIN ENDS
END START
```

Letter d.

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
   MOV AL, 5
   MOV BL, 4
   CMP AL, BL
   JG SUCCESS
   JBE FAIL

   MOV CL, 3
   CMP BL, CL
   JG SUCCESS
   JBE FAIL

SUCCESS:
   MOV DL, x
   MOV AH, 09H
   LEA DX, SUCCESSMESSAGE
   INT 21H

   INT 20H

FAIL:
   MOV AH, 09H
   LEA DX, FAILMESSAGE
   INT 21H

   INT 20H

SUCCESSMESSAGE DB "x= 1", 0DH, 0AH, "$"
FAILMESSAGE DB "Condition is not met", 0DH, 0AH, "$"
x DB 1

MAIN ENDS
END START
```

Letter e.

```
MAIN SEGMENT
ASSUME CS:MAIN, DS:MAIN, ES:MAIN, SS:MAIN
ORG 100h

START:
    MOV AX, 0
    MOV BX, 9

WHILE_LOOP:
    CMP AX, BX
    JGE END_WHILE


    MOV DL, AL
    ADD DL, '0'
    MOV AH, 02h
    INT 21h


    MOV DL, 0Dh
    INT 21h
    MOV DL, 0Ah
    INT 21h

    INC AX
    JMP WHILE_LOOP

END_WHILE:
    MOV AH, 09h
    LEA DX, FINAL_MESSAGE
    INT 21h

    ; Exit program
    MOV AH, 4Ch
    INT 21h

FINAL_MESSAGE DB "Loop ended at 9", 0Dh, 0Ah, "$"

MAIN ENDS
END START
```

PROBLEMS:

1. Assemble the following program and convert to a C program

```
MAIN SEGMENT

ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN

ORG 100h

START:

CALL MeBaby

CALL MeBaby

CALL MeBaby

CALL MeBaby

CALL MeBaby

CALL MeBaby

CALL MeBaby

INT 20h

MeBaby PROC

MOV AH,09

LEA DX,MSG

INT 21h

RET

MeBaby ENDP

MSG DB 'Assembly Programming is EASY! $'

MAIN ENDS

    END START
```

2. Convert the each of the following C codes into its equivalent assembly code:
   a. if (ebx<=ecx) { eax=5;edx=6;}
   b. if (var1<=var2) var3=15; else var3=10;var4=20;
   c. if (al>bl) && (bl=cl) x=1;
   d. if (al > bl) || (bl > cl) x=1;
   e. while (eax < ebx) eax =eax +1;

7. Conclusion:

The different syntax (requires or not) by assembly offer vast possibilities in generating results through directly talking to CPU and memory, The labels help both the assembler and the programmer navigate key parts of an assembly code. Loops help simplify repetitive instructions, and lastly, jump syntaxes significantly help conditional statements for various uses.

8. Assessment (Rubric for Laboratory Performance):