National University of Computer and Emerging Sciences

Laboratory Manual

for

Computer Organization and Assembly Language

Course Instructors

Lab Instructor(s)

Section

Semester

Department of Computer Science

COAL Lab 4 Manual

Objectives:

- Size of Program, Little Endian and Big Endian Order
- Data Segment
- Multiple Initializer/ARRAY
- Symbolic Constants
- Problems & Assignments

4.1 Size of program

To calculate the size of your program, open list file (.lst) of your program. From .lst file just find out starting and ending offset value of your program. To get size of your program find difference of ending and starting value of offset, this difference will show the size of your program in HEXABYTES.

For example, for some program, if Starting value of offset = 0100H & Ending value of offset = 0125H then,

Size of program is = Ending value of offset - Starting value of offset = 0125H – 0100H = 25H = 37d Bytes

4.1.1 Little Endian and Big Endian Orders:

Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored at the lowest storage address and least significant value in sequence is stored at highest storage address. While little-endian is an order in which the "little end" (least significant value in the sequence) is stored at lowest storage address while most significant value is stored at highest storage address.

Instruction	Big-endian Order		Little-endian Order	
	Address	Value	Address	Higher Address
VAR 12345678H	00 00 01 00	12	00 00 01 00	78
	00 00 01 01	34	00 00 01 01	56
	00 00 01 02	56	00 00 01 02	34
	00 00 01 03	78	00 00 01 03	12
VAR 1256, 8008, 1046	00 00 01 04	12	00 00 01 04	56
	00 00 01 05	56	00 00 01 05	12
	00 00 01 06	80	00 00 01 06	08

00 00 01 07	08	00 00 01 07	80

In list file (.lst) you can check how the data is saved in memory using little-endian order.

4.2 Data Segment:

It is also a large contagious chunk of memory in ram which is used for storing variables.

1. Directive is .data for data segment
2. All variables must be declared, and memory space for each allocated.
3. Data definition directive can be followed by a single value, or a list of values separated by commas.

Different *data definition* directives for different size types of memory are given below.

1. BYTE - Define Byte (8 bits)
2. SBYTE - Define Signed Byte (8 bits)
3. WORD - Define Word (16 bits)
4. SWORD - Define Signed Word (16 bits)
5. DWORD - Define Double Word (32 bits)
6. SDWORD - Define Signed Double Word (32 bits)
7. QWORD - Define Quad Word (64 bits)

4.3 MULTIPLE INITIALIZER/ARRAY:

An ARR is just consecutive sequence of memory bytes or words. For example, to define a three byte ARR called B_ARR, whose initial values are 10H, 20H, we can write.

```
B_ARR       BYTE  10H, 20H, 30H
```

The name B_ARR is associated with first of these bytes, B_ARR+1 with the second, B_ARR+2 with the third. If assembler assigns the offset address 0200H to B_ARR, then memory would look like this:

Symbol	Address	Contents
B_ARR	0200h	10h
B_ARR+1	0201h	20h
B_ARR+2	0202h	30h

In the same way, an ARR of words may be defined. For example;

```
W_ARR       WORD  1234H, 36H, 4568H, 502H
```

sets up any ARR of four words (8 bytes). If the ARR starts at 0300H, it will look like this:

Symbol	Address	Contents
W_ARR	0300h	1234

W_ARR+2	0302h	0036
W_ARR+4	0304h	4568

4.3.1 DUP directive

May be used to reserve more than one consecutive data item and initialize reserved items to same value. For example the instruction:

```
B_ARRAY    BYTE  100 DUP(0)
```

Instructs the assembler to reserve an array of 100 bytes and initialize each byte with zero value. In case of nested DUP, inner directive will be executed first.

```
B_ARRAY    BYTE  5 DUP (3 DUP(0))
```

4.3.2 Calculating size of array

Pseudo-op	Explanation	Syntax	Example
$	gives the address of location where used	VARIABLE DATA DEFINITION $	NUM DW $

```
B_ARRAY    BYTE  1, 2, 3, 4, 5

ARRsize    =     ($ - LIST)
```

1. ARRsize must follow immediately after LIST.
2. In case of Word array, length will become half of ARRsize.

4.4 SYMBOLIC CONSTANTS

A symbolic constant (or symbolic definition) is created by associating an identifier (or symbol) with an integer expression or some text. Some other properties of symbols are

1. Do not reserve storage
2. Cannot change at runtime

4.4.1 EQU

To assign a name to a constant, we use Equates directive.

1. No memory is allocated for EQU names.
2. Pseudo-ops (EQU) are not translated into machine code.
3. They simply tell the assembler to do something.
4. Do not allow redefinitions

Pseudo-op	Explanation	Syntax	Example
EQU	Use to assign a name to a constant	VARIABLE EQU CONSTANT VARIABLE EQU <TEXT> VARIABLE EQU EXPRESSION	Y EQU 8

4.4.2 EQUAL SIGN

It associates a symbol name with an integer expression.

Pseudo-op	Explanation	Syntax	Example
=	Use to assign a name to a constant/expression	NAME = CONSTANT	COUNT = 60H COUNT = 10H*10H

Some useful operators are as follows:

4.4.3 OFFSET Operator

Pseudo-op	Explanation	Syntax
OFFSET	Returns the offset of any data label	MOV DEST, OFFSET VARIABLE

```
.DATA
VAL      BYTE     10H
.CODE
MOV    AX, OFFSET VAL
```

4.4.4 PTR Operator

PTR operator overrides the default size for operand's address. It is useful when source and destination operand size are different.

Pseudo-op	Explanation	Syntax
PTR	It overrides the default size for operand's address.	MOV AL, BYTE PTR VARIABLE

```
.DATA
MYDOUBLE DWORD 12345678H
.CODE
MOV    AX,   MYDOUBLE                    ;ERROR!!
MOV    AX, WORD PTR MYDOUBLE             ;WORKS!
```

Problem(s) / Assignment(s)

Discussion & Practice **Estimated completion time: 1 hr, 30 mins**

Example 4.1: Assemble the given program and give answers the questions given below. **Estimated completion time:20 mins**

```
Lab4.asm
(Global Scope)
include Irvine32.inc
.data
B_VAR1 BYTE 14H
W_VAR2 WORD 2465H
D_VAR3 DWORD 1234567H
Y EQU 8
COUNT = 10H * 10H
NUM1 DWORD $
.code
main proc
MOV BX, W_VAR2
MOV DX, WORD PTR D_VAR3
ADD BL, BYTE PTR D_VAR3
MOV EAX, OFFSET B_VAR1
MOV ECX, NUM1
call DumpRegs
exit
main endp
end main
```

1. **Arrange the contents of memory location of D_VAR3 in memory?**

Sr.	Physical Address	Content
1	00406007	67
2	00406008	45
3	00406009	23
4	0040600A	01

2. **What does the value of NUM1 indicates?**

Num1 is constantsdfdsfe

3. **In instruction** `MOV DX, WORD PTR D_VAR3`**, why is there need of PTR operator?**

4. **Find out the offset address of W_ARR2 and COUNT from memory1?**

Example 4.2: Assemble the given program and give answers the questions given below. **Estimated completion time:20 mins**

Lab4.asm*

(Global Scope)

```
include Irvine32.inc
.data
B_ARRAY1 BYTE 10H, 20H, 30H, 40H
W_ARRAY2 WORD 2455H, 3478H, 98H
D_ARRAY3 DWORD 12345678H, 11236784H
NUM1 BYTE 2 DUP (3)
NUM2 BYTE 3 DUP (?)
.code
main proc
MOV AL, B_ARRAY1
MOV AH, B_ARRAY1+2
MOV DX, W_ARRAY2+1
MOV CL, BYTE PTR W_ARRAY2+3
MOV EBX, D_ARRAY3+2
MOV BX, WORD PTR D_ARRAY3+1
MOV NUM1+1, AH
MOV NUM2, CL
MOV NUM2+2, AL
call DumpRegs
exit
main endp
end main
```

1. **What is the value of BX register after instruction** `MOV BX, WORD PTR D_ARRAY3+1`**?**

2. After the execution of whole program write down the updated value of array NUM2 and NUM1?

Problem 4.1: *Arithmetic Expression* **Estimated completion time:15 mins**

Write a program that implements the following arithmetic expression.

EAX = (val1 + 7 + val2) + (-val1+val3)

Use the following data definition:

```
val1 SDWORD 8
val2 SDWORD -15
val3 SDWORD 20
```

In comments next to each instruction, write the hexadecimal value of EAX. Insert a call DumpRegs statement at the end of the program.

Problem 4.2: *Array Manipulation* **Estimated completion time:20 mins**

Insert the following variables in your program:

```
.data
Uarray WORD 1000h, 2000h, 3000h, 4000h
Sarray SWORD -1, -2, -3, -4
```

Write instructions that moves the four values in Uarray to the EAX, EBX, ECX, EDX registers. When you follow this with a call DumpRegs statement, the following register values should display:

EAX=00001000 EBX=00002000 ECX=00003000 EDX=00004000

Next, write instructions that moves the four values in Sarray to the EAX, EBX, ECX, EDX registers. When you follow this

with a call DumpRegs statement, the following register values
should display:

EAX=FFFFFFFF EBX=FFFFFFFE ECX=FFFFFFFD
EDX=FFFFFFFC

You are done with your exercise(s), make your submission ☺