Lab 3: Addressing Modes and I/O Operations

Objective

- To understand different addressing modes in Assembly language
- To learn about interrupts and Input/Output operations in Assembly language

Introduction

Addressing mode refers to the topic, that how we access different registers and data stored in them. Basically, we are talking about the ways to access data in register.

Activity Time boxing

Task No.	Activity Name	Activity time	Total Time
1	Lab Manual Lecture	60 mins	
2.	Example	10 mins	
3.	Walkthrough Tasks.	20 mins	
4.	Lab Tasks	60 mins	
5.	Evaluation	30 mins	180 mins

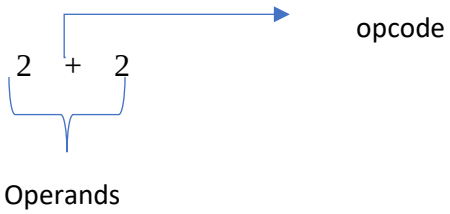
Concept Map

- Addressing Modes
- Mnemonics
- Interrupts
- Input/Output Operations

Addressing modes

Addressing mode refers to the topic, that how we access different registers and the data stored in them. Basically, we are talking about the ways to access data in register. Below is the example of the assembly language instruction. Let`s suppose 1st 2 is stored in register **dl** and another 2 is store in register **al** then the instructional command will be

ADD dl, [Address]



We need to store both the numbers somewhere so that we can add them. When both the data is placed in the register then we call it **registers addressing (**when both operands are register**)**.

If we have this kind of instructional command like **ADD dl, 2.** If we have one operand saved in the register and another operand is constant. This type of addressing is called **immediate addressing (**where one operand is constant**).**

Let`s suppose we have the following kind of instruction:

ADD al, [address] where we have one operand saved in the register and another operand is saved in the memory (RAM). This kind of addressing is called **memory addressing** (where one operand is from memory).

These are the three types of accessing and saving data into registers. CPU use this type of addressing to access data and in assembly language.

```
Add dl, dh        ;     Register Addressing
Add dl, 2         ;     Immediate Addressing
Add dl, [address]:      Memory Addressing
```

Mnemonics

mnemonic is a symbolic name for a single executable machine **language** instruction.

Mnemonics

These are the mnemonics that are commonly used in assembly language:

Mnemonic	Description
LDR	Load from memory location into a register
ADD	Add data in a register
SUB	Subtract data in a register
STR	Store values from register into a memory location
B	Branch to a marked position in the program
CMP	Compare values in a register
AND	Bitwise logical AND operation
ORR	Bitwise logical OR operation
XOR	Bitwise logical XOR operation
MVN	Bitwise logical NOT operation
LSL	Logically shift the value left, used for multiplication
LSR	Logically shift the value right, used for division
HALT	Stops the program

Interrupt

An interrupt is an event that causes the processor to suspend its present task and transfer control to a new program called the interrupt service routine (ISR).

The 8086 INT instruction generates a software interrupt. For I/O and some other operations, the number used is 21.

A specific number is placed in the register AH to specify which I/O operation (e.g., read a character, display a character) you wish to carry out.

When the I/O operation is finished, the interrupt service program terminates, and program will be resumed at the instruction following int.

Example: Write a code fragment to display the character 'a' on the screen

```
;To Display the character 'a'
org 100h
.data
.code
main proc
mov dl, 'a'
mov ah,2h
INT 21H
main endp
end main
```

Character Input from User

To get the input from keyboard a subprogram at **1h** will be called. First **1h** will be placed in **ah** and then an interrupt **21h** generated to call the subprogram. Finally, the character will be placed in **al** register.

Example

```
mov ah,1h       ; Keyboard input subprogram
INT 21H         ; Call the subprogram to get character input and stored in al
```

If we run the following commands like

MOV dl, 2

MOV ah, 2

The first command will store the data to the data register and second command will then print the data in the data register. In order to print the data on the screen, CPU need to access the screen and it has to stop and access the screen and display the data on the screen, for this purpose we use interrupt to stop the CPU and make to do something for us and then resume its normal tasks. In simplified words the interrupts can be used to stop the program and allow microprocessor to access hardware to take input or give output. Two types of interrupts we will be using in assembly language are

INT 21h = interrupt for text handling

INT 20H = interrupt for video/graphics handling

For example

For output

```
MOV ah, 2
INT 21h
```

For input

```
MOV ah, 1
INT 21h
```

ASCII code

American standard code for information interchange.

If we run the below code to display the character 2 on the screen

```
For output
MOV dl, 2
MOV ah, 2
INT 21h
```

After this code we expect to get the character 2 on the screen but we will get ☺ or something else. It will not display the character 2 on the screen, some other character will be printed based on the ASCII code. ASCII is the standard encoding scheme, because we have CPUs from different vendors, therefore, a common standard is being established for different characters that are entered from the keyboard.

As we discussed that 2 will not be displayed on the screen, instead of it ☺ will be displayed because 2 is the ASCII for☺. We have to learn ASCII code instead of the original data to be displayed on the screen. Below is the sequence of ASCII for different type of data

A = 65, B = 66 → Z = 90

a = 97, b = 98 → z = 122

0 = 48, 1 = 49 → 9 = 57 0

Now if we want to print 2 on the screen, we will update our above code as follows:

```
For output
MOV dl, 50
MOV ah, 2
INT 21h
```

Instead of writing 2. We are now using 50 which is and ASCII code for 2. Now on the screen we will get the number 2 instead of ☺

Another two important ASCII codes

10 = for next line

13 = carriage return

Both these commands are used to start from the next line.

ASCII CODE TABLE

0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	"	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	"	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	'	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	'	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	°
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	fi	254	˛
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	fl	255	ˇ

Practice Task

Practice Tasks will be available in separate document and will be uploaded on Moellim in relevant week.

Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Further Reading

The slides and reading material can be accessed from the folder of the class instructor available at Moellim.

Outcomes

The outcomes of this lab were:

1. Understanding of Addressing Modes in Assembly Language
2. Usage of I/O Operations in Assembly Language