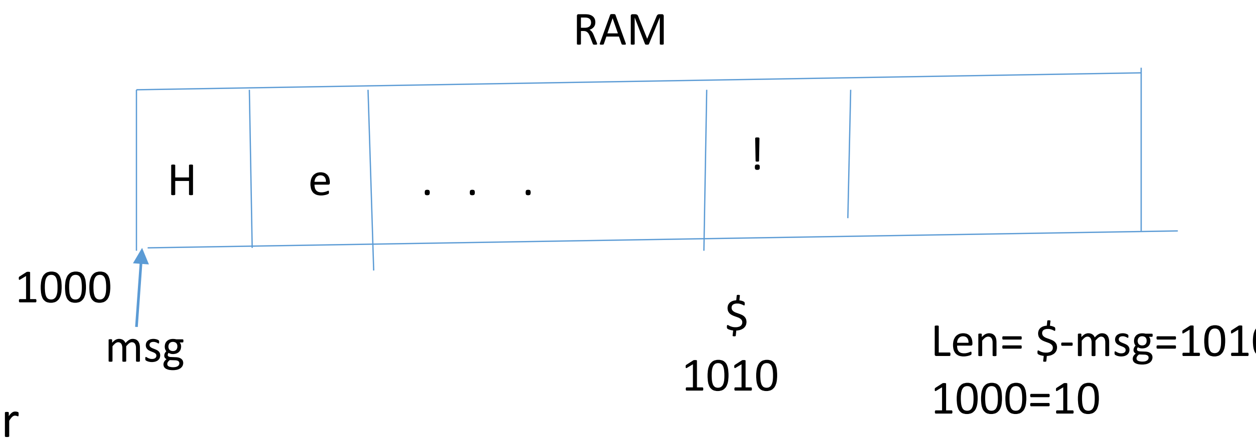
Assignment No 1

Assignment no 1

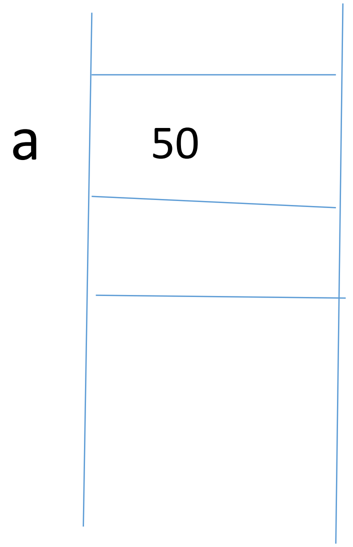
- Write an ALP for 64 bit Arithmetic operations and display the result. Accept the numbers from the user.

Hello world program

```
segment .data                    ;data segment
msg db "Hello, world!",10          ; msg to display
Len equ $ - msg                    ;length of our string

segment .text                    ;code segment
global  _start                 ;must be declared for linker
_start:                        ;tell linker entry point

mov rax,1              ;system call number (sys_write)
mov rdi,1              ;file descriptor (stdout)
mov rsi,msg            ; message to write
mov rdx,len            ; message length
Syscall                ;call kernel
mov rax,60             ;system call number (sys_exit) (60-for exit)
mov rdi,0   ; to return 0 value like return 0
Syscall                ; call kernel
```



Int a=50 ;



Ideone online platform for assembly language program execution
https://ideone.com/FHyc1A

Table 1.1. Standard Files Provided by Unix

Descriptive Name	Short Name	File Number	Description
Standard In	stdin	0	Input from the keyboard
Standard Out	stdout	1	Output to the console
Standard Error	stderr	2	Error output to the console

Command to install nasm

$whereis nasm

If showng path nasm is already installed if not execute following command

$sudo dnf install nasm ; for fedora

$ sudo apt-get install nasm ; for ubuntu edit fork download copy

```
segment .data                                  ;data segment
msg db 'Hello, world!',10             ;     msg to display
len equ $ - msg                                  ;length of our string
segment .text                                  ;code segment
global  _start                                 ;must be declared for linker
_start:                                        ;tell linker entry point

mov rax,1                            ;system call number (sys_write)
mov rdi,1                             ;file descriptor (stdout)
mov rsi,msg                         ; message to write
mov rdx,len                       ; message length
Syscall                                 ;call kernel
mov rax,60                         ;system call number (sys_exit) (60-for exit)
Mov rdi,0
syscall; call kernel
```

Success #stdin #stdout 0s 5524KB comments (?)

stdin copy

```
Standard input is empty
```

stdout copy

```
Hello, world!
```


https://ideone.com/FHyc1A

language: Assembler 64bit (nasm 2.14)
created: 0 seconds ago
visibility: secret

Share or Embed source code

<script src="https://ideone.com/e.js/FHyc1A" type="text/javascript" ></script>

To install Nasm,Open fedora terminal, type su to enter into root

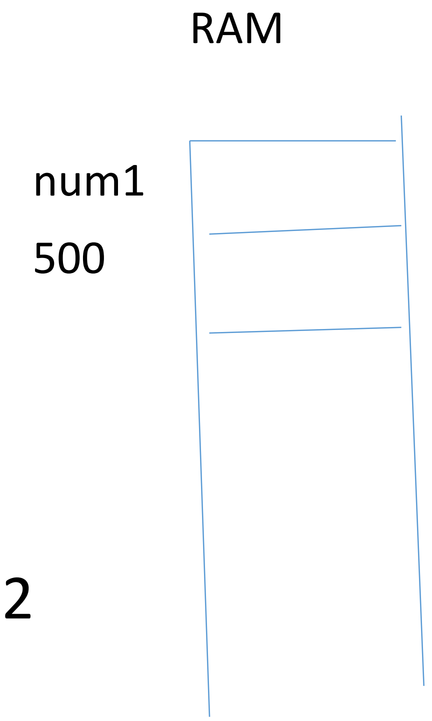
Then type

dnf install nasm on terminal

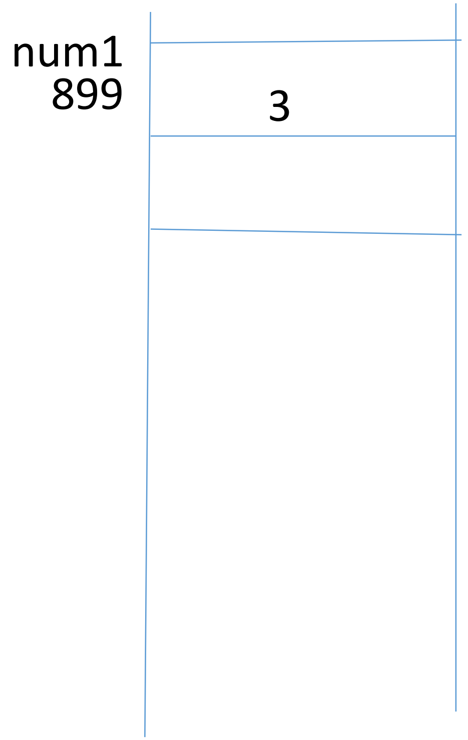
It will install nasm

Write an ALP (64 bit) for addition of two numbers by accepting inputs through keyboard

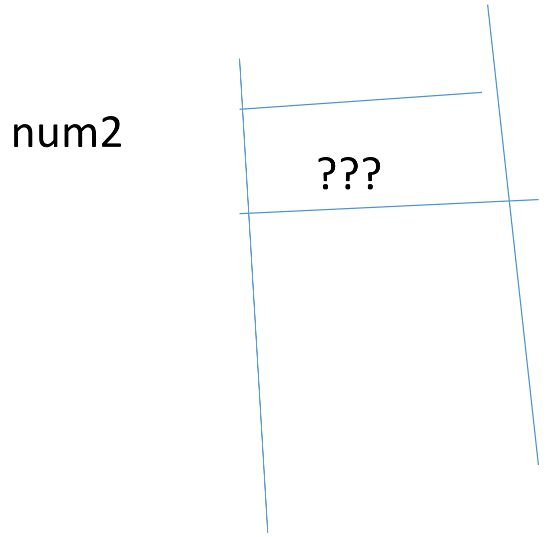
```
section .data
msg db "Enter first number",10     ; Enter first no
len equ $-msg                      ; length of first number
msg1 db "Enter second number",10   ; Enter second no
len1 equ $-msg1                    ; length of second number
msg2 db "The sum is“,10
len2 equ $-msg2

section .bss
num1 resb 2 ; allocate memory for first number like int num1
num2 resb 2 ; Allocate memory for second number like int num2
sum resb 2      ; Allocate memory for sum like int sum
```



```
section .text
global _start
_start:
mov rax,1        ; write operation/display first message
mov rdi,1 ; fd value monitor
mov rsi,msg
mov rdx,len
syscall

mov rax,0        ; read/input first number
mov rdi,0 ;fd value of the keyboard
mov rsi,num1
mov rdx,2
syscall
```



```
mov rax,1       ; display second message
mov rdi,1
mov rsi,msg1
mov rdx,len1
syscall

mov rax,0       ; input second number
mov rdi,0
mov rsi,num2
mov rdx,2 ;size of num2
syscall
```



https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html

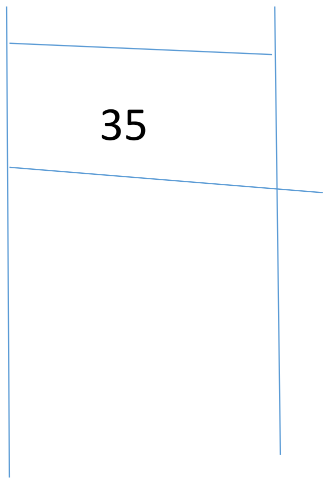
```
mov rax,[num1]  ; move first number in rax
sub rax,30h  ; convert it in original number
mov rbx,[num2] ; move second number in rbx
sub rbx,30h ; convert it in original number

add rax,rbx  ; add two numbers
add rax,30h ;convert it in ascii
mov [sum],rax ; mov result in sum

mov rax,1      ;display sum
mov rdi,1
mov rsi,msg2
mov rdx,len2
syscall
```

For example nem1=2 is 32 H & num2=3 is 33H

So 2+3=5

& 32+33=65H is which is wrong

sum



ASCII Code | ASCII Table | ASCII Character Set

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
mov rax,1   ; display sum
mov rdi,1
mov rsi,sum
mov rdx,2
syscall

mov rax,60 ; end system call
mov rdi,0
syscall
```

Compiling and Linking an Assembly Program in NASM

1. Type the code using a text editor(gedit) and save it as filename.asm.
2. Make sure that you are in the same directory as where you saved filemane **.asm**.
3. To assemble the program, type **nasm -f elf64 hello.asm [elf64-> executable and linking file for 64 bit]**

If there is any error, you will be prompted about that at this stage. Otherwise an object file of your program named **filename.o** will be created.

To link the object file and create an executable file named hello, type

4. **ld -o filename filename.o (ld-load)**

5. Execute the program by typing **./filename** on terminal prompt.

Assembly - Procedures

- Procedures or subroutines are very important in assembly language/HLL, as the programs tend to be large in size.
- Procedures are identified by its name, like person.
- Following this procedure name, the body of the procedure is described which performs a well-defined job/specific task.
- End of the procedure is indicated by a **return** (ret) statement.

- As programs get larger and larger, it becomes necessary to divide them into a series of procedures.

• A procedure is a block of logically-related instruction that can be called by the main program or another procedure at any place for any times.

• Each procedure should have a single purpose and be able to do its job independent of the rest of the program.

Advantages of procedure

1) Reusability
2) Reduce complexity
3) Chances of error are reduced

Following is the syntax to define a procedure −

proc_name:

procedure body

..

ret

- The procedure is called from another function by using the **CALL instruction**.
- The CALL instruction should have the name of the called procedure as an argument as shown below −

Call proc_name

- The called procedure returns the control to the calling program/ procedure by using the RET instruction.

CMP instruction

- The **CMP instruction** compares two operands.

- It is generally used in **conditional execution**.

- This **instruction** basically subtracts one operand from the other for comparing whether the operands are equal or not.

- It does not disturb the destination or source operands.

- Syntax

  CMP destination, source

- The destination operand could be either in register or in memory.
- The source operand could be a constant (immediate) data, register or memory.

JBE instruction (Jump **Below/Equal** or Jump Not Above)

- JBE Jump If Below or Equal Flags: Jump if CF = 1 or ZF = 1 Used **after a CMP or SUB instruction.**
- JBE transfers control to short- label if the first operand is **less than or equal to the second. (if a $\le$ b)**

- Else no action is taken.
- Syntax : JBE dest (lable)
- Example:

CMP AX, 0030H; compares by subtracting 0030H from the value in AX register

JBE LABEL1; jumps to the address specified by LABEL1 if value in register AX is ;below or equal to the value 0030H (if (ax $\le$ 0030H) is true)

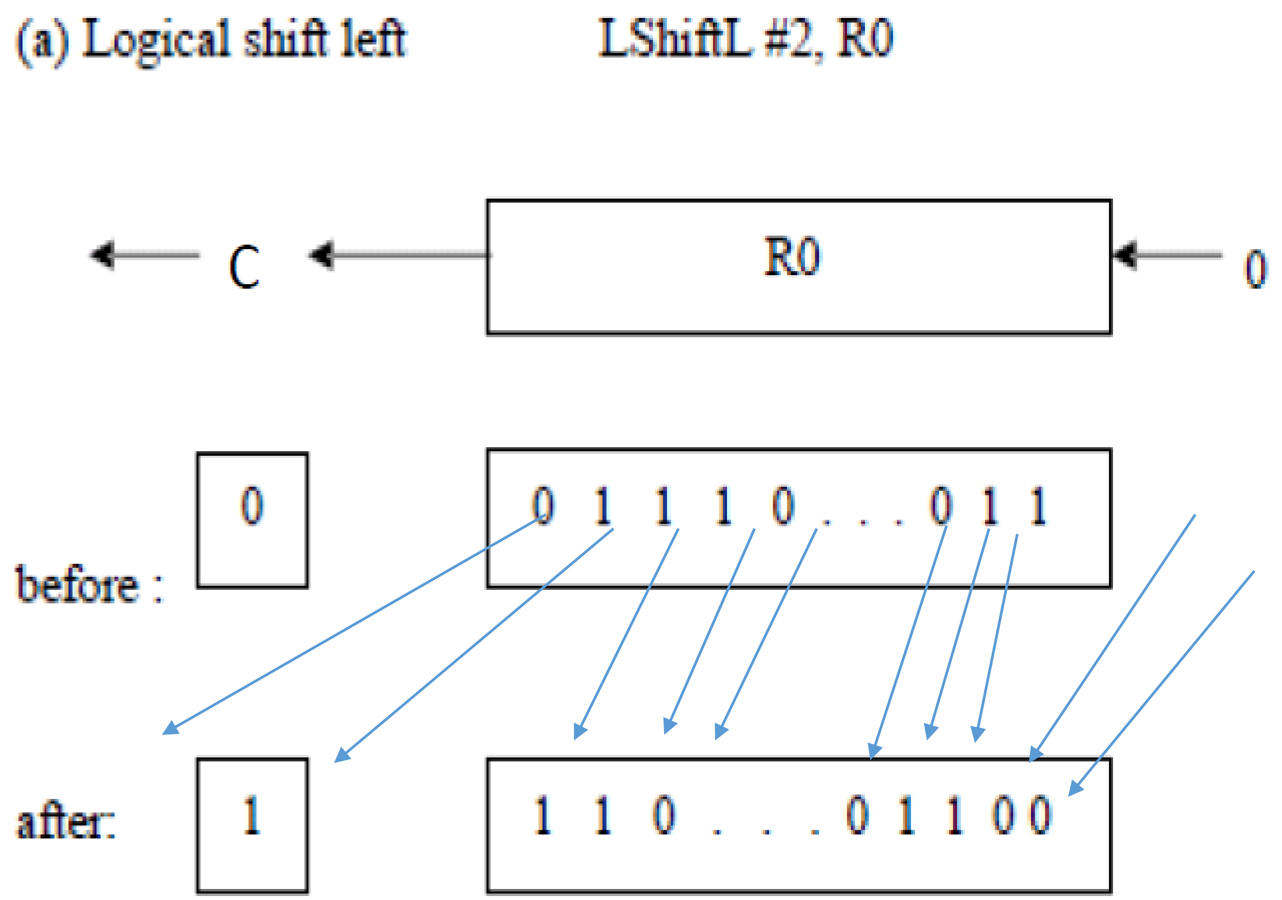
The SHL (shift left) instruction

- It performs a logical left shift on the destination operand, filling the lowest bit with 0 i.e. LSB.

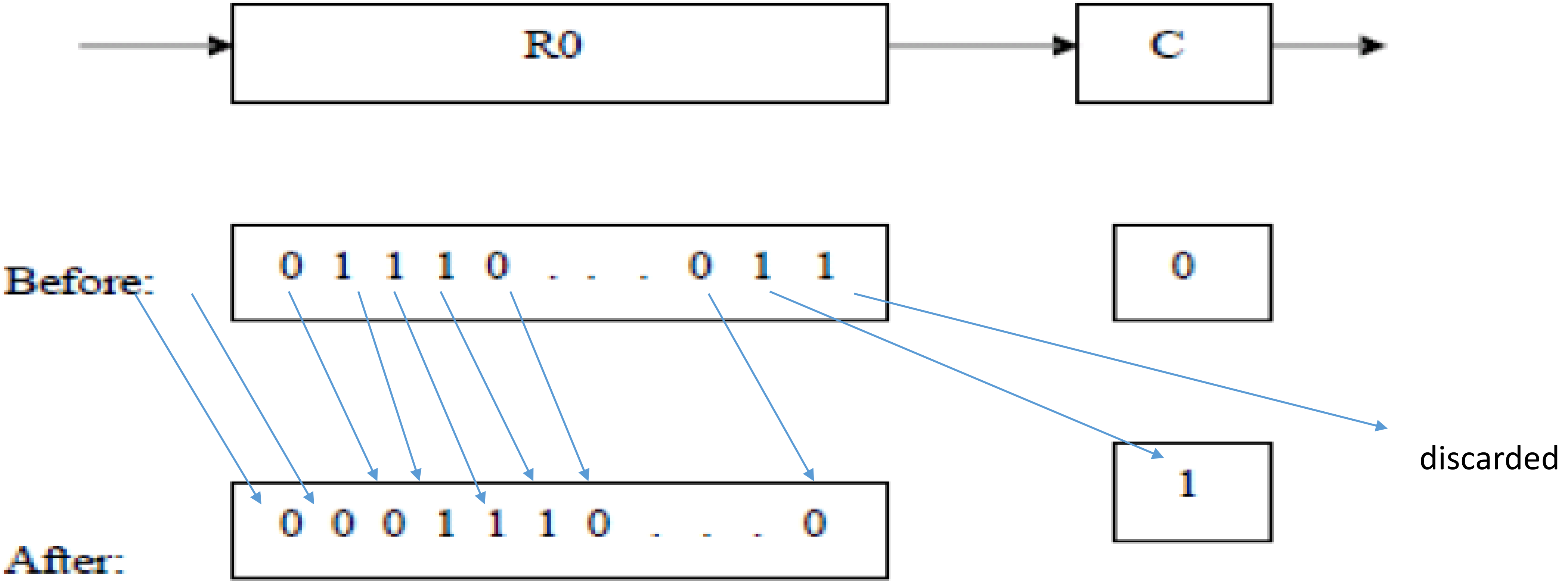
- The SHR (shift right) instruction performs a logical right shift on the destination operand.
- The highest bit position is filled with a zero (MSB).

- Shifts the bits in the first operand (destination operand) to the left or right by the number of bits specified in the **second operand** (count operand).

- Bits shifted beyond the destination operand boundary are first shifted into the **CF flag**, then discarded.
- At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.
- The destination operand can be a register or a memory location.
- The count operand can be an immediate value or register.


(a) Logical shift left
LShiftL #2, R0
C
R0
0
before :
0
0 1 1 1 0 . . . 0 1 1
after:
1
1 1 0 . . . 0 1 1 0 0

(b) Logical shift right LShiftR #2, R0



The jnz (or jne) jump if non zero instruction

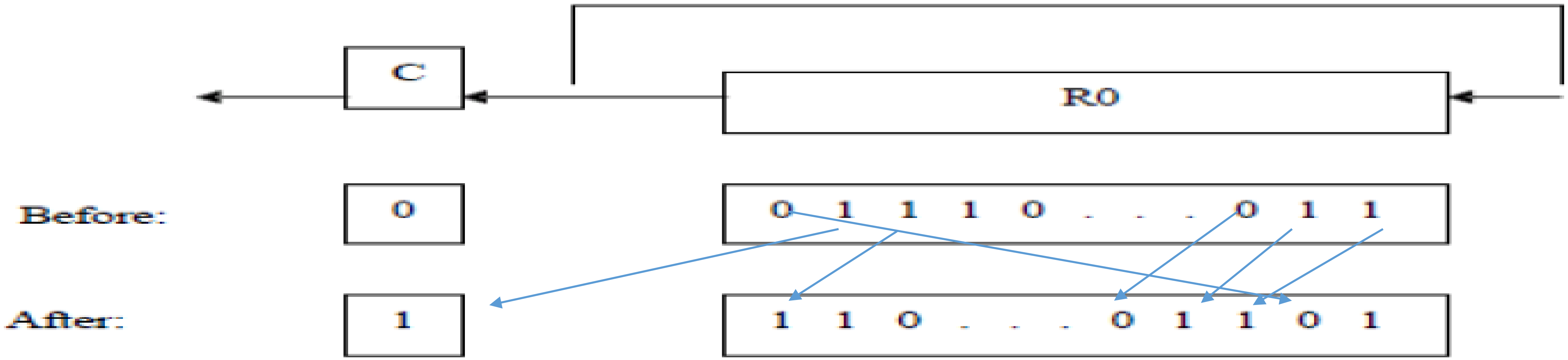
- It is a conditional jump that follows a test.
- It jumps to the specified location if the **Zero Flag (ZF) is cleared (0).**
- **jnz** is commonly used to explicitly test for something not being equal to zero whereas jnz is commonly found after a **cmp instruction**.

JNE/JNZ	Jump not Equal or Jump Not Zero	ZF

Rol (rotate left) Instruction

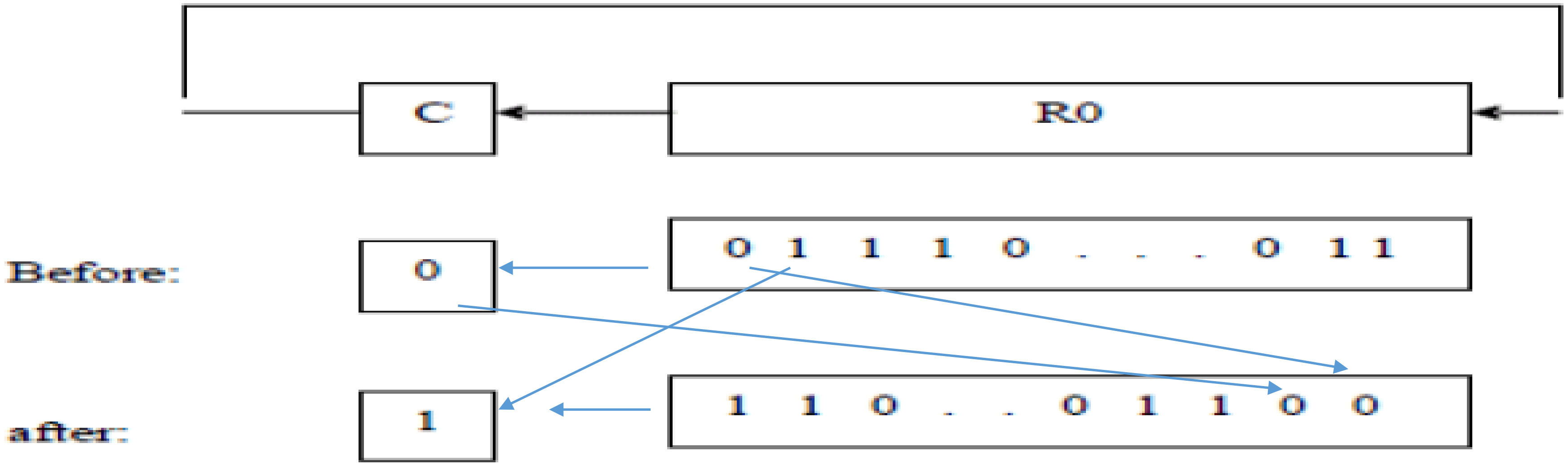
- The left rotate **instruction** shifts all bits in the register or memory operand specified.
- The most significant bit is rotated to the carry flag, the carry flag is rotated to the least significant bit position, all other bits are shifted to the left.

(a) Rotate left without carry RotateL #2, R0



rol

(b) Rotate left with carry RotateLC #2, R0



And instruction

- This instruction perform the specified logical operation (logical bitwise and) on their operands, placing the result in the first operand location.

- The AND instruction is used for supporting logical expressions by performing bitwise AND operation.

- The bitwise AND operation returns 1, if the matching bits from both the operands are 1, otherwise it returns 0.
- For example **and bl,0F H** let bl =CC

CC -1100 1100
OF -0000 1111
0000 11 00 (0C)

Syntax

and <reg>,<reg>
and <reg>,<mem>
and <mem>,<reg>
and <reg>,<con>
and <mem>,<con>

https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

- The macro is invoked by using the macro name along with the necessary parameters. When you need to use some sequence of instructions many times in a program, you can put those instructions in a macro and use it instead of writing the instructions all the time.

- Writing a macro is another way of ensuring modular programming in assembly language.

- A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.

- In NASM, macros are defined with %macro and %endmacro directives.

- The macro begins with the %macro directive and ends with the %endmacro directive.

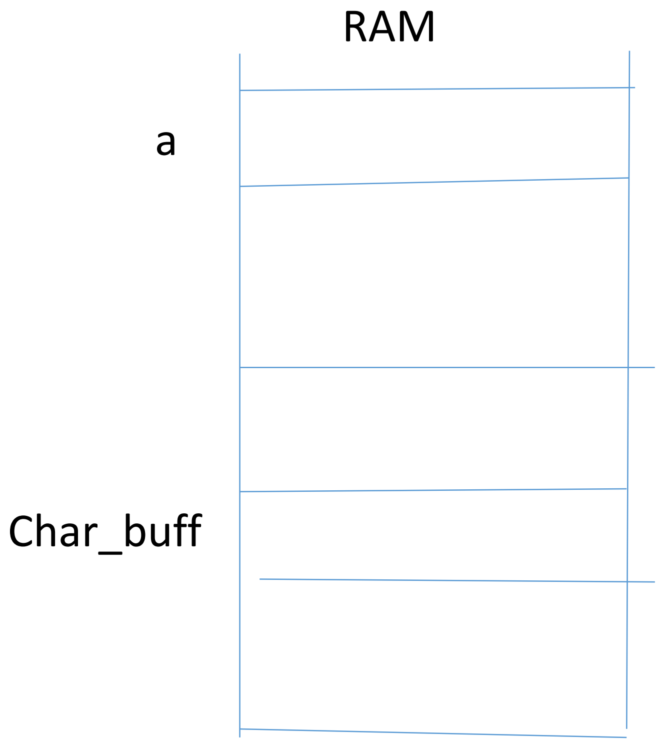
The Syntax for macro definition −

```
%macro macro_name,  number_of_params
<macro body>
%endmacro
```

Where, number_of_params specifies the number parameters, macro_name specifies the name of the macro.

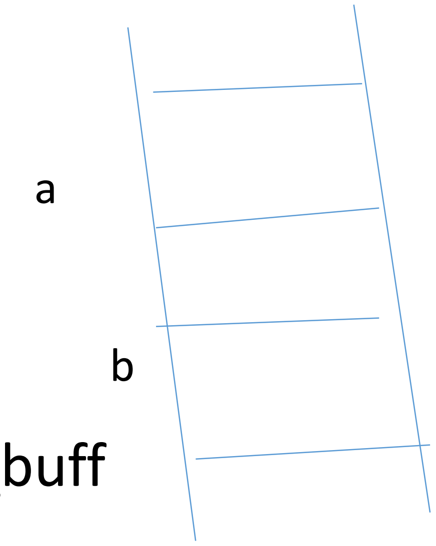
```
%macro READ 2
mov rax,0
mov rdi,0
mov rsi,%1
mov rdx,%2 ; two parameters
syscall
%endmacro

%macro WRITE 2
mov rax,1
mov rdi,1
mov rsi,%1
mov rdx,%2
syscall
%endmacro
```

```
section .data
msg1 db "Enter first number",10
len1 equ $-msg1
msg2 db "Enter second number",10
len2 equ $-msg2
```




```
section .bss
a resq 1            ;reserve 1 quadword(64bit) memory for first number
b resq 1            ;reserve 1 quadword(64 bit)memory for second number
char_buff  resb 16 ; reserve 16 byte memory for sum (4 bits/digit i.e. (16 digit))
```

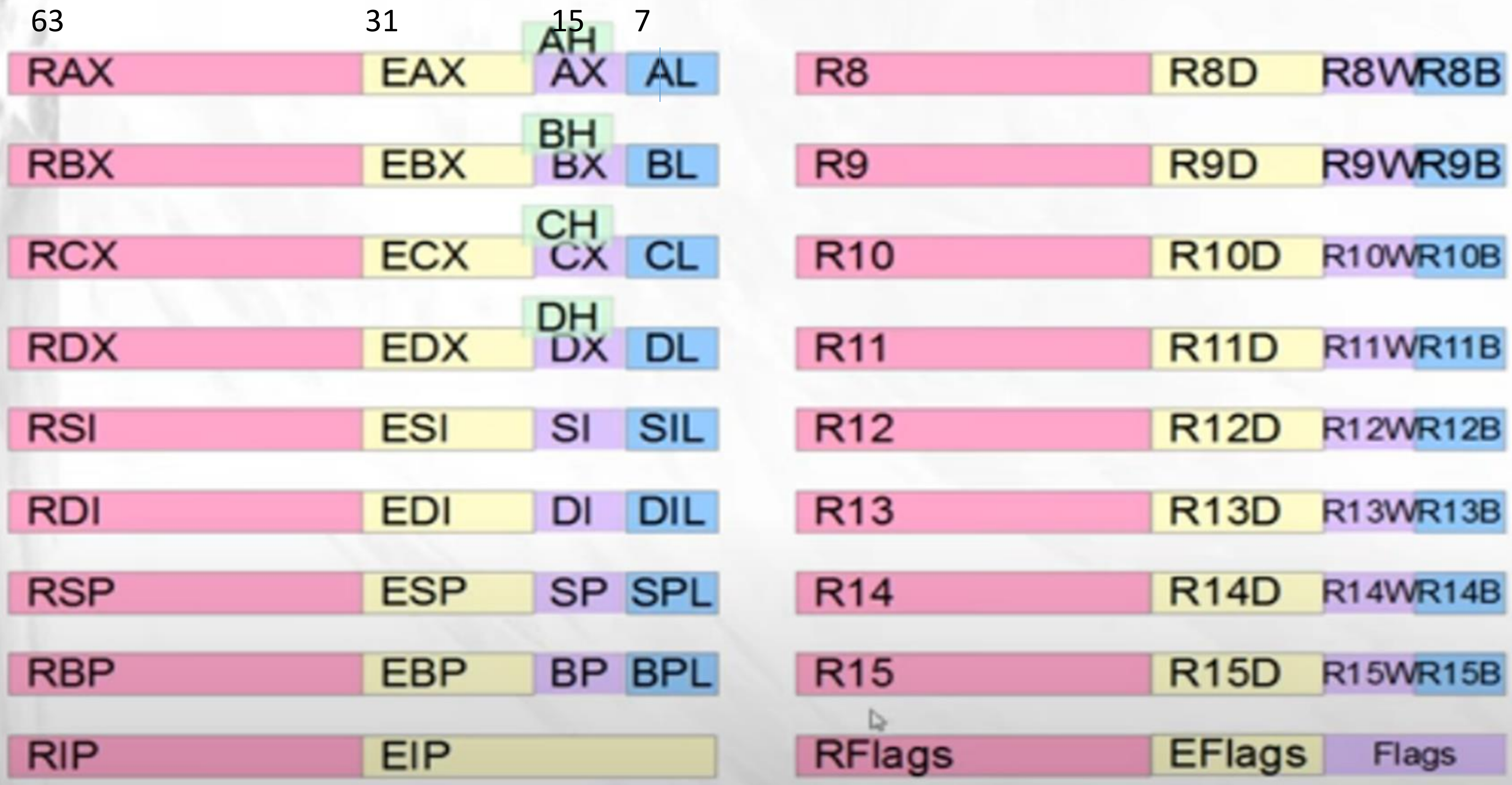
```
section .text
global _start:
_start:

WRITE msg1,len1   ;write micro, display first message

READ char_buff,17  ; read macro,read first number and store in Char_buff (16+1 for enter)

dec rax              ;when we input number total length is stored in rax. Total 16 digit number is
; equal to 64 bit number .when 16 digit number is entered and then enter is pressed. Total 17
;bytes are stored in rax. We have entered in 16 digit. So decrement contents of rax.

mov rcx, rax  ; number of digit counter

call accept   ; call accept procedure to covert entered number (ascii) into original 16 digit
                         ;number
```

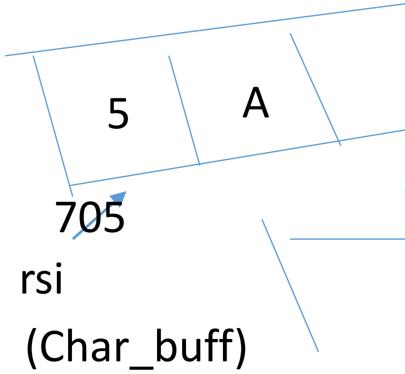



```
mov qword[a],rbx   ; store  64 bit number in variable a

WRITE msg2,len2      ;display second number
READ char_buff,17    ; read second number and store in Char_buff

dec rax                  ; decrement contents of rax

mov rcx,rax         ; transfer rax value in rcx

call accept      ; call accept procedure to covert entered number (ascii) into original 16
                  ;digit number

mov qword[b],rbx    ; store  16 digit number in variable b
```

```
mov rbx,qword[a]    ; move first number in rbx

add rbx,qword[b]      ;add two numbers

call display     ; call display procedure to display result

mov rax,60 ; Exit system call
mov rdi,0
Syscall
```

Pentium IV and x64






```
accept:
        mov rsi,char_buff     ;rsi pointing to first digit of entered number
        mov rbx,00            ;clear the contents of rbx to store inputted digit
      up:mov rdx,00  ;clear the contents of rdx
        mov dl,byte[rsi]      ;mov first digit(one byte )to dl register
        cmp dl,39H            ;compare dl with 39h
        jbe sub30             ; if less than or equal to 39 then subtract 30 so go to label subroutine30.
        sub dl,07H      ; if not less or equal to 39 then subtract 7
```

rcx=16

```
sub30:sub dl,30H
         shl rbx,04           ;shift contents of rbx by one digit to left in hex 4 bits are required
         add rbx,rdx          ; add contents of rbx with rdx
         inc rsi  ; inc rsi pointer
        dec rcx    ;decrement counter value
        jnz up    ; repeat this procedure until rcx value is non zero.
         ret  ; return back to calling procedure.
```

```
display:                                   ;Display procedure to display result
      mov rcx,16      ; number is 16 digit ,is considered as counter
      mov rsi, char_buff ; point rsi to char_buff

    up1:rol rbx,04  ;rotate contents of rbx by one digit (4 bits)msb shifted to lsb
      mov dl,bl       ;mov two digit which was in bl to dl
      and dl,0FH      ; convert to two digit to one digit using and instruction(0000 1111-0FH)
      cmp dl,09h   ; compare this dl with 09h (1 to 9 no), if dl≤ 09)
      jbe add30     ; jump below/equal if dl is less than or equal to 09 go to label add30
      add dl,07H    ; add 07h in dl

   add30:add dl,30H ; add 30h in dl
       mov byte[rsi],dl ; mov this dl in to memory pointing by rsi
       inc rsi      ; increment rsi
       dec rcx  ;decrement counter
       jnz up1   ;repeat this until counter is zero (jump if non zero)
    WRITE char_buff, 16      ;display the result present in char_buff
     ret  ; return back to calling procedure
```

CC -11001100
+ OF-0000 1111
0000 11 00 (0C

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

9

7