MIT World Peace University

Digital Electronics and Computer Architecture
Second Year B. Tech, Semester 3

Write an Assembly Language Program to Display 2 Digit and 4 Digit Hexadecimal Numbers using 64 Bit Assembly.

Practical Report
Assignment 7

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

December 3, 2022

Contents

1 Problem Statement

Write an assembly language program (ALP) to display 2-digit and 4-digit hex numbers using 64-bit assembly language programming.

2 Objective

- To understand the structure of the assembly language program.
- To understand system call function for write and exit.

3 Platform

CPU - Core i7 Duo, 64 bit with 4 GHz clock frequency.
OS - Arch Linux, 64 bit
Editor - VS code
Assembler - NASM (Netwide Assembler)
Linker - LD, GNU linker.

4 Theory

4.1 Assembly Language Program Basic Structure

```
section.Data (declare data segment)
; initialized data declaration
section.bss (declare block started by segment sort)
; uninitialized data declaration.
section.text (declare code segment)
global-start (entry point for program)

-start:
;code.

(;semicolon is used to give the comment)
```

4.2 System calls to write and exit.

4.2.1 System Call to write/output call

display variable -name contents of specified variable length on monitor.

```
mov rax,1   ; function number for writing/outputting the data.
mov rdi,0   ; file descriptor ID for standard input device (keyboard)
mov rsi,arr ; starting addresses of the variable used to store the data.
mov rdx,8   ; maximum bytes to be read.
syscall     ; system call (in built function)
```

4.2.2 System exit call

function to exit or terminate program.

```
mov rax,60 ; function number for sys-exit
mov rdi,0  ; return code for zero error.
syscall    ; system call.
```

4.3 Instruction used in the program for implementation

```
ADD
add two numbers together

COMPARE
compare numbers

JUMP
jump to designated RAM address.

LOAD
Load information from RAM to the CPU.
```

4.4 Commands to execute the program

```
to assemble:
nasm -f elf64 hello.asm

to link:
ld -o hello hello.o

to execute:
./hello

where, hello is the filename.
```

5 Algorithm

1. start
2. Display message "Two-digit HEX Number"
3. Initialize hard coded two digit and four-digit number.
4. Write a procedure for unpacking BCD number (Display Ouputs)
5. Display two digit and four digit numbers.
6. end.

6 Input

Two digit and four digit numbers.

7 Output

Two digit and four digit numbers.

```
The Two digit Hex number is:
2A
```

8 Code

```
; display 2 digit hexx numbers
section .data
    msg db "The Two digit Hex number is: ", 10
    msglen equ $-msg
    num1 db 2AH ; the number to be printed. h is for hex

section .bss
; temp data assignment
    sum resb 1
    temp resb 1

section .text
global _start

_start:
    ; printing the first message
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, msglen
    syscall

    ; assign one byte of num1 to al
    mov al, byte[num1]
    ;assign the value of al to sum
    mov byte[sum], al
    ;assign 2 to bp
    mov bp, 2; bp = 2
    ; shift all binary bits 4 times to right, this flips the nibbles
    ; so rn its 0010 0011 after flipping it becomes 0011 0010

up:rol al, 4
    ; assign al to bl
    mov bl, al; al = 32H
    ; and with 0FH, so 0000 1111 anded with 0000 0010 so youll end up with the
    0010
    and al, 0FH ; al = 02H at this point
    ; this would trigger some flag
    cmp al, 09
    ; goto down label if the above cmp statement gives less than or equal to
    jbe down
    add al, 07H
```

```
42
43 down: Add al, 30H; al = 32H
44     mov byte[temp], al
45     mov rax, 1
46     mov rdi, 1
47     mov rsi, temp
48     mov rdx, 1
49     syscall
50     mov al, bl ; bl = 23H
51     dec bp ; this is the loop register which we decrement if its 0 then we stop
       the loop
52     jnz up; now go to up again, and this time you would use bls value to al coz
       you would rotate it again.
53
54 mov rax, 60
55 mov rdi, 0
56 syscall
```

9 Conclusion

Thus, implemented the program in assembly language to display two digit and four-digit hex numbers

10 FAQs

1. **Explain assembler directives. List the assembler directives in your program.**

   Assembler Directives supply data to the program and control the assembly process. It enables to do the following:

   - Assemble code and data into specified sections.
   - Reserve space in memory for unitized variables.
   - Control the appearance of listings.
   - Initialize memory.
   - Assemble conditional blocks.
   - Define global variables.
   - Specify libraries from which the assembler can obtain macros.
   - Examine symbolic debugging information.

   Assembler Directives in the program:

   - .text switch to text segment.
   - .data switch to initialized part of data segment.
   - .bss switch to uninitialized port of data segment.

2. **Illustrate the significance of the sections: .data, .bss, .text** .bss segment stands for 'block starting symbol' is the memory space for uninitialized variable of your code . IT is the method of optimization to reduce the code size.
   syntax: section.bss
   var-name RES memory-Type memory size.

```
    Eg. section.bss
        A resb 5D
        (Declare variable A allocate 50 bytes memory)
```

.data section holds the initialized value. It holds the data of the initialized variable (global or local)

```
syntax:
    section.data
    var_name data_type variable_value.
Eg:
    A DB 50
    (declared variable A of type byte with value 50)
```

.text segment is the code, vector table and constants. It is the section that holds the executable instructions.

```
syntax:
       section.text
       global_start
       _start:
       ;code
```

3. **What is the difference between RESB and DB?**

   DB stands for Declare/Define Byte. It is a directive that is used to allocate space for initialized data in the data section.

   RESB Stands for Reserve Byte. It is a directive that is used to allocate space for uninitialized data in .bss section.