COA Lab Assignment 3

Roll No.: CS20B1016

Name: V Nagasai

Q1. By taking one example program (i.e., any program done in labs so far), write the ALPs for the following addressing modes

1. **Direct Addressing Mode.**
2. **Indirect Addressing Mode.**
3. **Immediate Addressing Mode.**
4. **Direct-Offset Addressing Mode.**
5. **Relative Addressing Mode.**
6. **Indexed Addressing Mode.**
7. **Register Addressing Mode.**
8. **Register Indirect Addressing Mode.**
9. **Auto-Increment Addressing Mode.**
10. **Auto-Decrement Addressing Mode.**

Solution:

```
section .data
    msg db "The sum is: "
    mlen equ $ - msg
    nl db 0xa
    nlen equ $ - nl
    num1 dq 10
    num2 dq 20
    arr dq 10, 20

section .bss
    num3 resb 4

section .text
    global _start

direct:
    mov eax, [num1]
    add [num2], eax
    mov eax, [num2]
    ret
```

```
indirect:
    mov dword [num3], num2  ; set num3 as the address of num2
    mov eax, [num3]         ; mov the content (stored address)
    mov ebx, [eax]          ; add to ebx, the content form the address stored in eax
    add [num1], ebx
    mov eax, [num1]
    ret

immediate:
    add dword [num1], 20
    mov eax, [num1]
    ret

direct_offset:
    mov eax, [arr]
    add eax, [arr + 8]
    ret

relative:
    mov dword [num3], num2
    mov ecx, num3
    mov ebx, [num3]
    mov eax, [ebx]
    add eax, [num1] ;eax=[ebx]=[num3]=address of num2
    ret

indexed:
    mov ecx, arr
    mov eax, [ecx]
    add eax, [ecx + 8]
    ret

register:
  mov eax, [num1]
  mov ebx, [num2]
  add eax, ebx
    ret

register_indirect:
  mov eax, [num1]
  mov ebx, num2
  add eax, [ebx]
    ret

auto_increment:
    mov ebx, 8
    mov ecx, arr
    mov eax, [ecx]

    auto_inc:
        add eax, [ecx + ebx]
        add ebx, 8
        cmp ebx, 32
        jne auto_inc
    ret

auto_decrement:
    mov ebx, 16
    mov ecx, arr
    mov eax, [ecx + 24]

    auto_dec:
        add eax, [ecx + ebx]
        sub ebx, 8
        cmp ebx, -8
        jne auto_dec
    ret

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, mlen
    int 0x80

select_mode:
    ; call direct
    ; call indirect
    ; call immediate
    ; call direct_offset
    ; call relative
    ; call indexed
    ; call register
```

```
    ; call register_indirect
    ; call auto_increment
    call auto_decrement

;----------------------------------------
; Integer printing function (itoa)
print_int:
    push    eax             ; preserve eax on the stack to be restored after function runs
    push    ecx             ; preserve ecx on the stack to be restored after function runs
    push    edx             ; preserve edx on the stack to be restored after function runs
    push    esi             ; preserve esi on the stack to be restored after function runs
    mov     ecx, 0          ; counter of how many place holders we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    idiv    esi             ; divide eax by esi
    add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide instruction
    push    edx             ; push edx (string representation of an intger) onto the stack
    cmp     eax, 0          ; can the integer be divided anymore?
    jnz     divideLoop      ; jump if not zero to the label divideLoop

printLoop:
    dec     ecx             ; count down each byte that we put on the stack
    mov     eax, esp        ; mov the stack pointer into eax for printing
    call    print_str       ; call our string print function
    pop     eax             ; remove last character from the stack to move esp forward
    cmp     ecx, 0
    jnz     printLoop       ; jump is not zero to the label printLoop

    pop     esi             ; restore esi from the value we pushed onto the stack at the start
    pop     edx             ; restore edx from the value we pushed onto the stack at the start
    pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
    pop     eax             ; restore eax from the value we pushed onto the stack at the start
    ret


str_len:
    push    ebx             ; preserve ebx on the stack to be restored after function runs
    mov     ebx, eax        ; move eax into ebx

nextchar:
    cmp     byte [eax], 0   ; compare the byte pointed to by EAX at this address against zero (Zero is an end of string delimiter)
    jz      finished        ; jump (if the zero flagged has been set) to the point in the code labeled 'finished'
    inc     eax             ; increment the address in EAX by one byte (if the zero flagged has NOT been set)
    jmp     nextchar        ; jump to the point in the code labeled 'nextchar'

finished:
    sub     eax, ebx        ; subtract the address in EBX from the address in EAX
    pop     ebx             ; Restore the original value of ebx before our function was called
    ret


print_str:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    str_len         ; call the str_len length function to get length of passed message

    mov     edx, eax        ; store the string's lenth in edx for printing
    pop     eax             ; Restore the original value of eax before our function was called

    mov     ecx, eax
    mov     ebx, 1          ; STDOUT
    mov     eax, 4          ; SYS_WRITE
    int     80h

    pop     ebx
    pop     ecx
    pop     edx
    ret



exit:
    call print_int

    mov eax, 4
    mov ebx, 1
    mov ecx, nl
    mov edx, nlen
    int 0x80
```

```
mov eax, 1
int 0x80
```