香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 06: Introduction to Assembly Language

Yuhong LIANG

yhliang@cse.cuhk.edu.hk

Outline

- Data Movement Instructions
- Arithmetic and Logic Instructions
- Control Flow Instructions

Data Movement Instructions

mov

- Syntax:

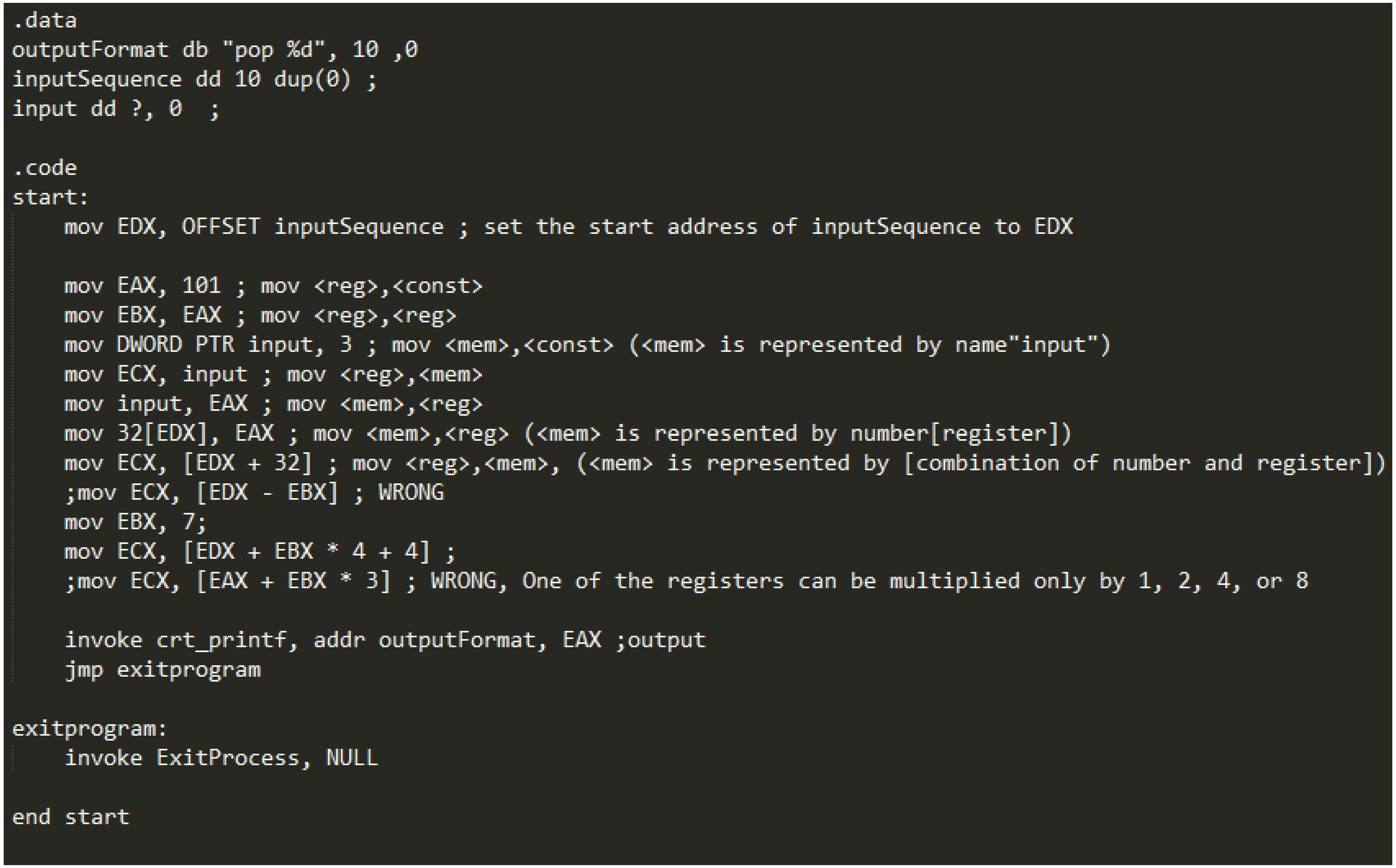
```
mov <reg>,<con>
mov <reg>,<reg>
mov <mem>,<con>
mov <reg>,<mem>
mov <mem>,<reg>
```

- Semantics:

The mov instruction moves the data represented by second label (i.e.register contents, memory contents, or a constant value) into the location represented by first label (i.e. a register or memory).

- <reg>：usually EAX,EBX,ECX,EDX…
- <con>:-1,0,1,2,3,4…
- <mem>:name, number[register], [combination of number and register]…

Data Movement Instructions



```
.data
outputFormat db "pop %d", 10 ,0
inputSequence dd 10 dup(0) ;
input dd ?, 0  ;

.code
start:
    mov EDX, OFFSET inputSequence ; set the start address of inputSequence to EDX

    mov EAX, 101 ; mov <reg>,<const>
    mov EBX, EAX ; mov <reg>,<reg>
    mov DWORD PTR input, 3 ; mov <mem>,<const> (<mem> is represented by name"input")
    mov ECX, input ; mov <reg>,<mem>
    mov input, EAX ; mov <mem>,<reg>
    mov 32[EDX], EAX ; mov <mem>,<reg> (<mem> is represented by number[register])
    mov ECX, [EDX + 32] ; mov <reg>,<mem>, (<mem> is represented by [combination of number and register])
    ;mov ECX, [EDX - EBX] ; WRONG
    mov EBX, 7;
    mov ECX, [EDX + EBX * 4 + 4] ;
    ;mov ECX, [EAX + EBX * 3] ; WRONG, One of the registers can be multiplied only by 1, 2, 4, or 8

    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Arithmetic and Logic Instructions

add, sub

- Syntax:

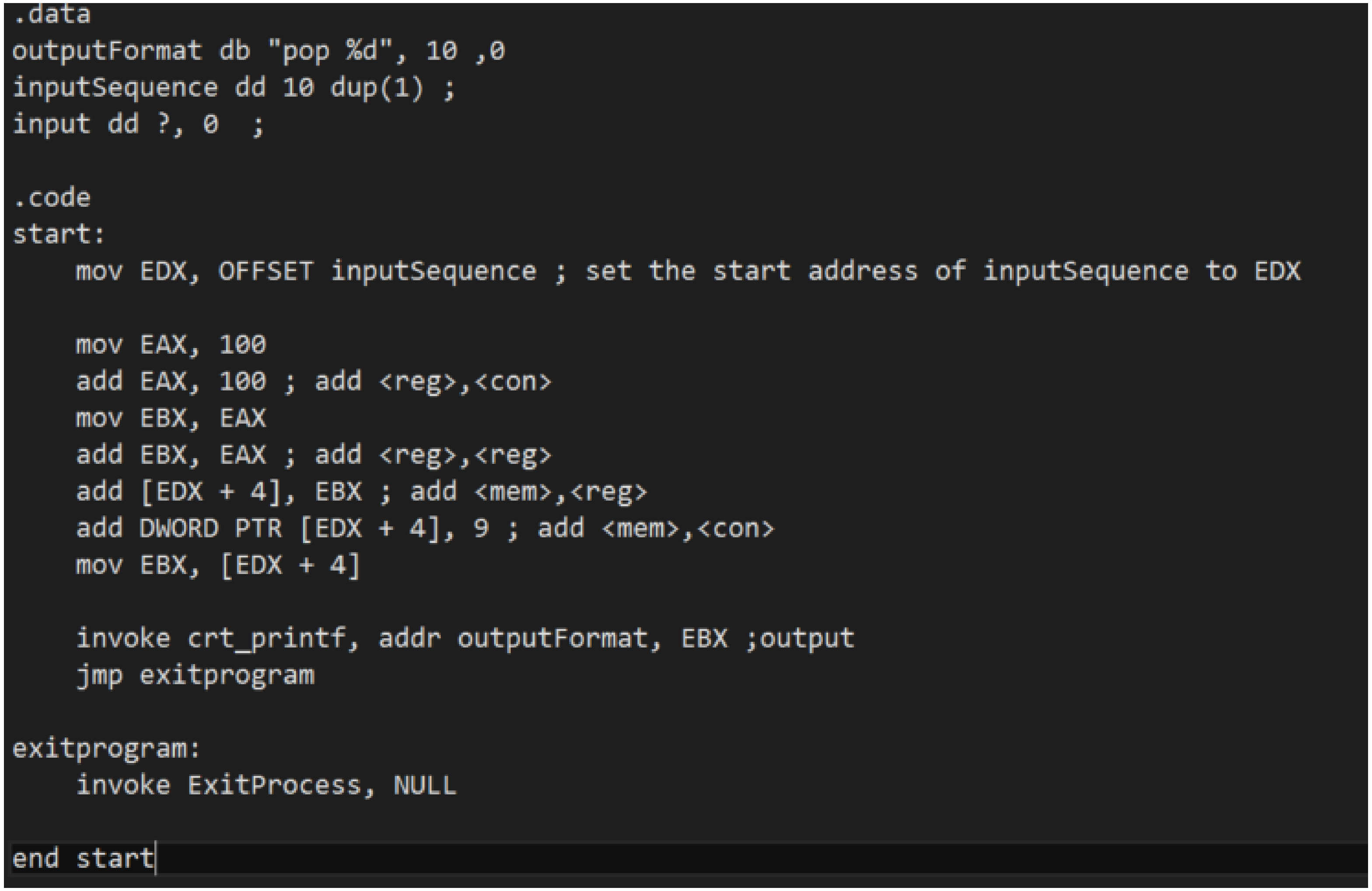
add <reg>,<reg>	sub <reg>,<reg>
add <reg>,<mem>	sub <reg>,<mem>
add <mem>,<reg>	sub <mem>,<reg>
add <reg>,<con>	sub <reg>,<con>
add <mem>,<con>	sub <mem>,<con>

- Semantics:

The add instruction adds together the data, storing the result in the location represented by its first label. Similarly, the sub instruction subtracts data represented by second label from the first.

Arithmetic and Logic Instructions

add, sub



```
.data
outputFormat db "pop %d", 10 ,0
inputSequence dd 10 dup(1) ;
input dd ?, 0  ;

.code
start:
    mov EDX, OFFSET inputSequence ; set the start address of inputSequence to EDX

    mov EAX, 100
    add EAX, 100 ; add <reg>,<con>
    mov EBX, EAX
    add EBX, EAX ; add <reg>,<reg>
    add [EDX + 4], EBX ; add <mem>,<reg>
    add DWORD PTR [EDX + 4], 9 ; add <mem>,<con>
    mov EBX, [EDX + 4]

    invoke crt_printf, addr outputFormat, EBX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Arithmetic and Logic Instructions

inc, dec

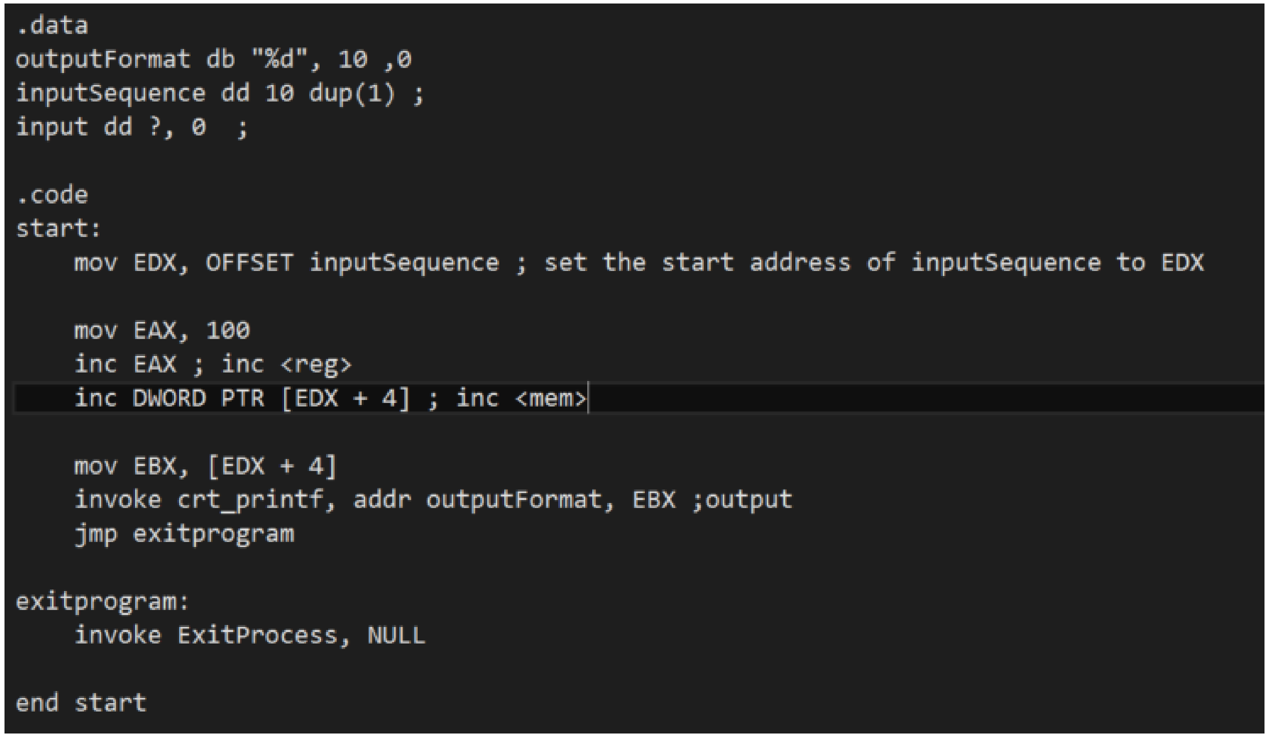
- Syntax:

inc <reg>   dec <reg>

inc <mem> dec <mem>

- Semantics:

The inc instruction increments the data represented by label by one, and similarly dec decrements the data represented by label by one



```
.data
outputFormat db "%d", 10 ,0
inputSequence dd 10 dup(1) ;
input dd ?, 0  ;

.code
start:
    mov EDX, OFFSET inputSequence ; set the start address of inputSequence to EDX

    mov EAX, 100
    inc EAX ; inc <reg>
    inc DWORD PTR [EDX + 4] ; inc <mem>

    mov EBX, [EDX + 4]
    invoke crt_printf, addr outputFormat, EBX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Arithmetic and Logic Instructions

imul

- Syntax:

imul <reg>,<reg>

imul <reg>,<mem>

imul <reg>,<reg>,<con>

imul <reg>,<mem>,<con>

- Semantics:

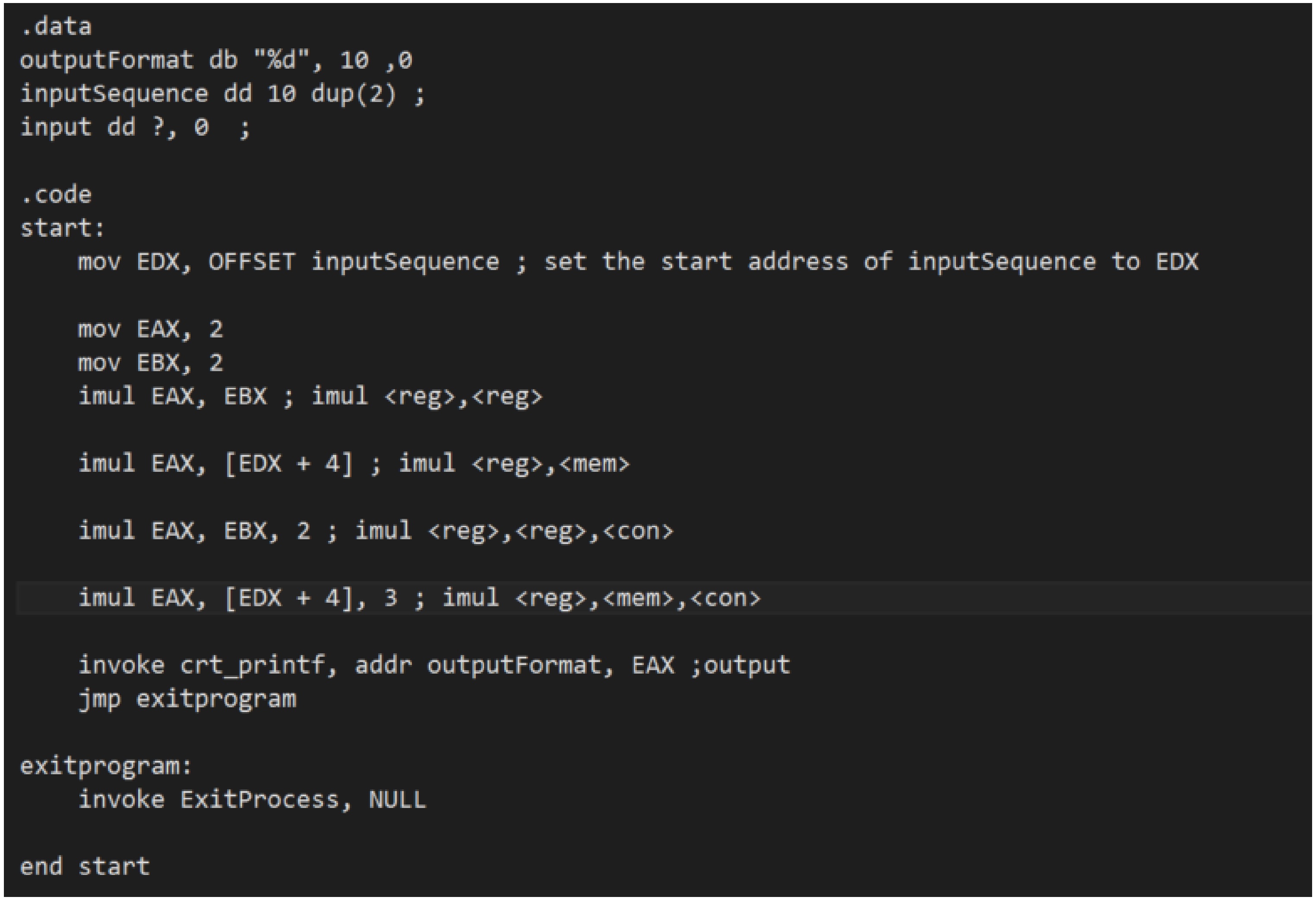
The imul instruction has two basic formats: two-operand and three-operand.

The two-operand form multiplies the data together and stores the result in location represented by by the first label.

The three operand form multiplies the data represented by its second and third label together and stores the result in location represented by its first label. Furthermore, the third operand is restricted to being a constant value.

Arithmetic and Logic Instructions

imul



```
.data
outputFormat db "%d", 10 ,0
inputSequence dd 10 dup(2) ;
input dd ?, 0  ;

.code
start:
    mov EDX, OFFSET inputSequence ; set the start address of inputSequence to EDX

    mov EAX, 2
    mov EBX, 2
    imul EAX, EBX ; imul <reg>,<reg>

    imul EAX, [EDX + 4] ; imul <reg>,<mem>

    imul EAX, EBX, 2 ; imul <reg>,<reg>,<con>

    imul EAX, [EDX + 4], 3 ; imul <reg>,<mem>,<con>

    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Arithmetic and Logic Instructions

idiv

- Syntax:

idiv <reg>

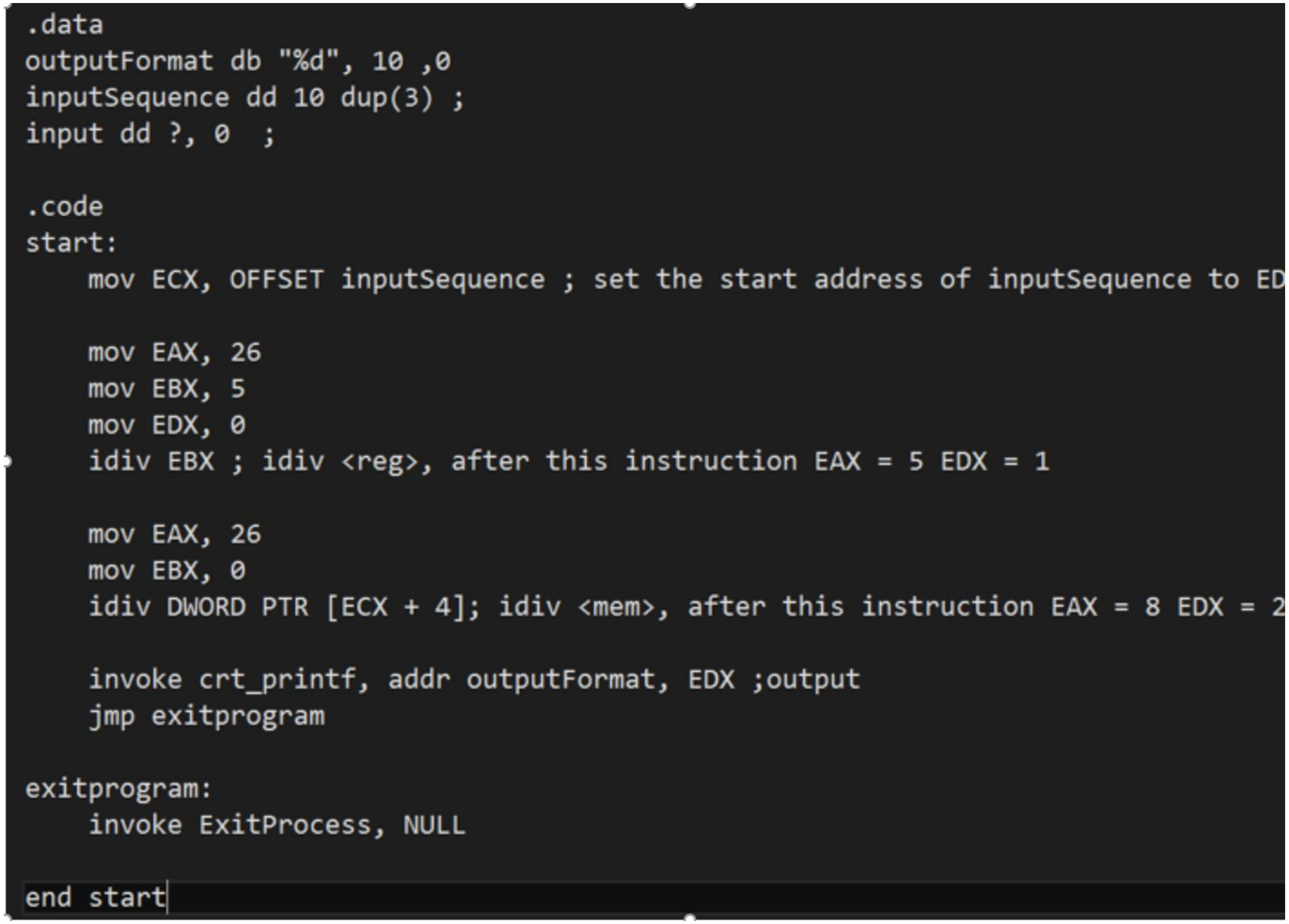
idiv <mem>

- Semantics:

The idiv instruction is used to divide the contents of the 64 bit integer EDX:EAX (constructed by viewing EDX as the most significant four bytes and EAX as the least significant four bytes) by the specified operand value. The quotient result of the division is stored into EAX, while the remainder is placed in EDX.

Arithmetic and Logic Instructions

idiv



```
.data
outputFormat db "%d", 10 ,0
inputSequence dd 10 dup(3) ;
input dd ?, 0  ;

.code
start:
    mov ECX, OFFSET inputSequence ; set the start address of inputSequence to ED

    mov EAX, 26
    mov EBX, 5
    mov EDX, 0
    idiv EBX ; idiv <reg>, after this instruction EAX = 5 EDX = 1

    mov EAX, 26
    mov EBX, 0
    idiv DWORD PTR [ECX + 4]; idiv <mem>, after this instruction EAX = 8 EDX = 2

    invoke crt_printf, addr outputFormat, EDX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Arithmetic and Logic Instructions

and, or

- Syntax:

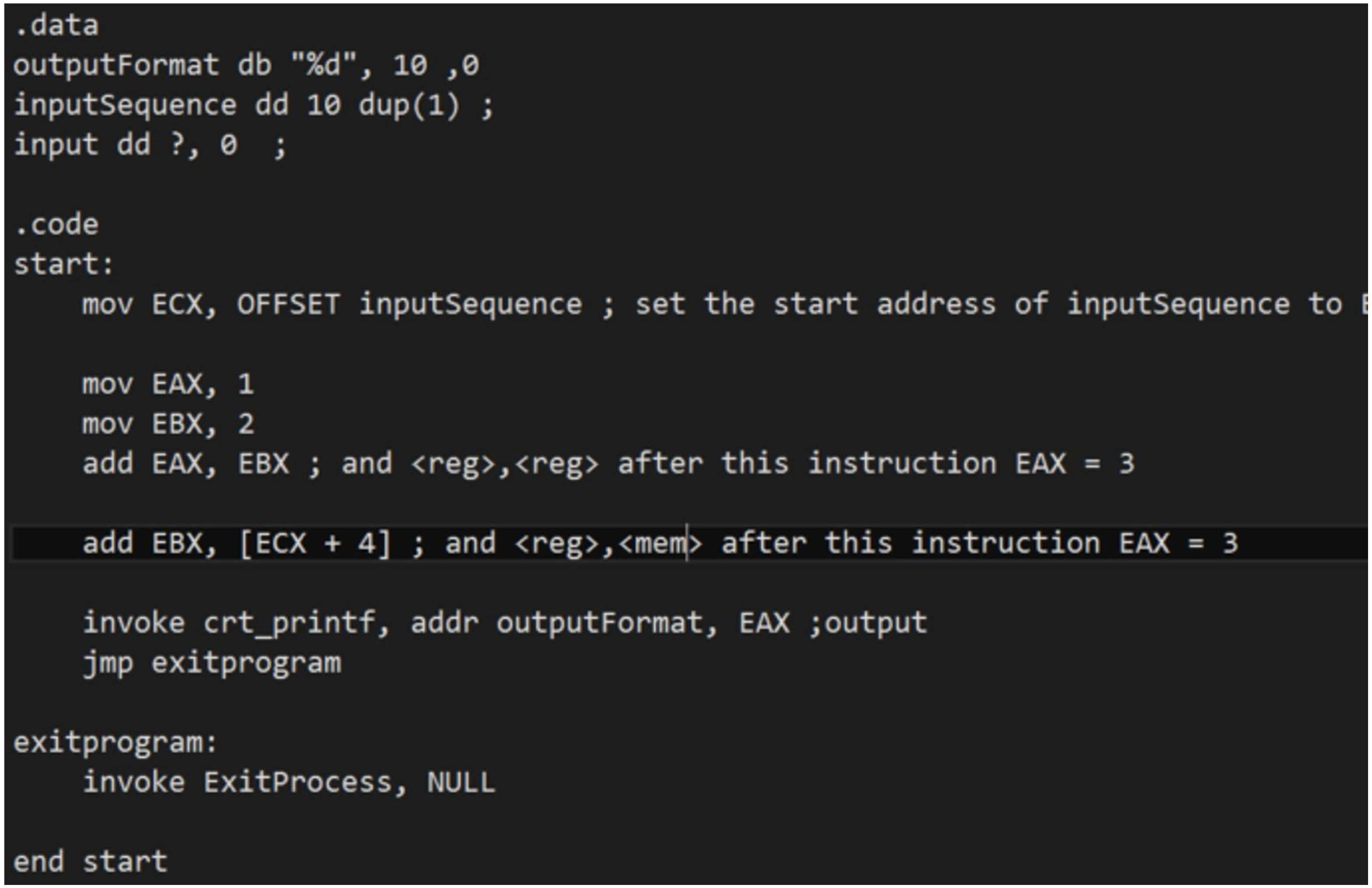
and <reg>,<reg>   or <reg>,<reg>
and <reg>,<mem> or <reg>,<mem>
and <mem>,<reg> or <mem>,<reg>
and <reg>,<con>   or <reg>,<con>
and <mem>,<con> or <mem>,<con>

Semantics:

These instructions perform the specified logical operation (logical bitwise and, or,
and exclusive or, respectively) on data, placing the result in location represented by the first
label.

Arithmetic and Logic Instructions

and, or



```
.data
outputFormat db "%d", 10 ,0
inputSequence dd 10 dup(1) ;
input dd ?, 0  ;

.code
start:
    mov ECX, OFFSET inputSequence ; set the start address of inputSequence to E

    mov EAX, 1
    mov EBX, 2
    add EAX, EBX ; and <reg>,<reg> after this instruction EAX = 3

    add EBX, [ECX + 4] ; and <reg>,<mem> after this instruction EAX = 3

    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram

exitprogram:
    invoke ExitProcess, NULL

end start
```

Control Flow Instructions

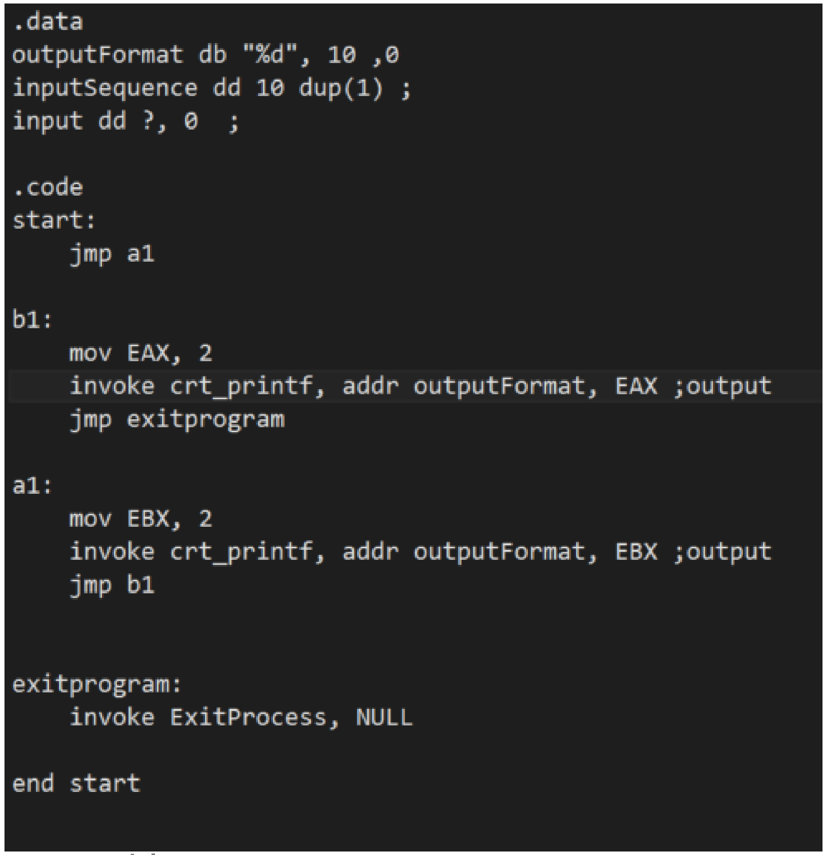
jmp

- Syntax:

jmp <label>

- Semantics:

Transfers program control flow to the instruction at the memory location indicated by the label.



```
.data
outputFormat db "%d", 10 ,0
inputSequence dd 10 dup(1) ;
input dd ?, 0  ;

.code
start:
    jmp a1

b1:
    mov EAX, 2
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram

a1:
    mov EBX, 2
    invoke crt_printf, addr outputFormat, EBX ;output
    jmp b1


exitprogram:
    invoke ExitProcess, NULL

end start
```

Control Flow Instructions

conditional jumps

- Syntax:

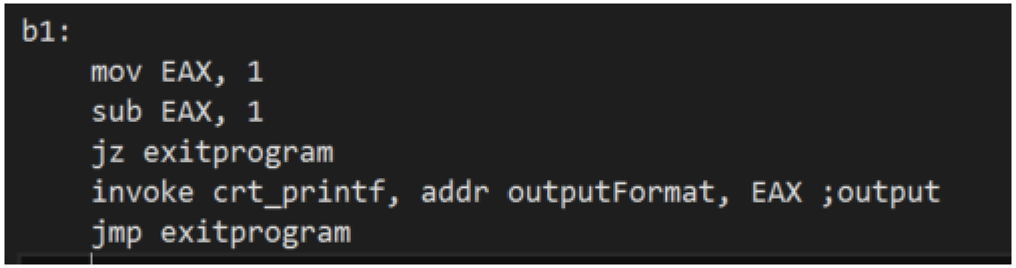
jz <label> - Jump when the zero flag is set

jo <label> - - Jump when the overflow flag is set

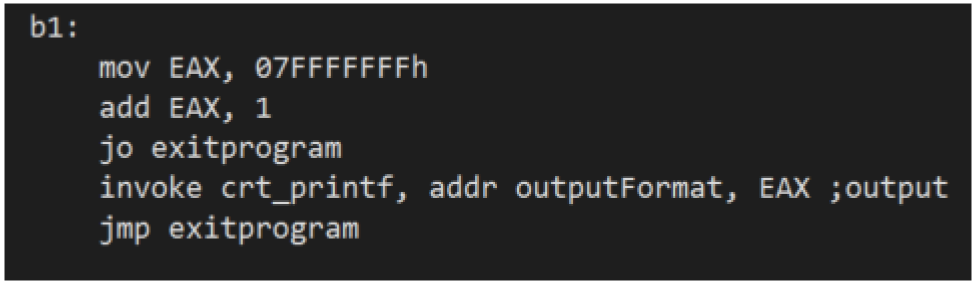
jc <label> -- Jump when the carry flag is set

- Semantics:

These instructions are conditional jumps that are based on the status of a set of condition flags that are stored in a special register. The flags in register is information about result of the last arithmetic instruction.



```
b1:
    mov EAX, 1
    sub EAX, 1
    jz exitprogram
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram
```



```
b1:
    mov EAX, 07FFFFFFFh
    add EAX, 1
    jo exitprogram
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram
```

Control Flow Instructions

conditional jumps

- Syntax:

je <label> - Jump when equal

jne <label> - Jump when not equal

jg <label> - Jump when greater than

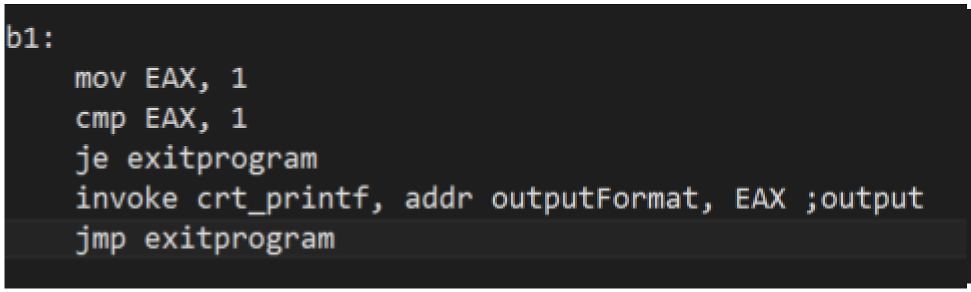
jge <label> - Jump when greater than or equal to

jl <label> - Jump when less than

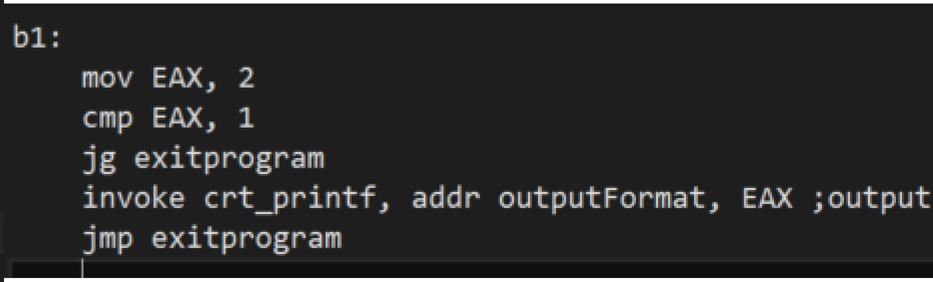
jle <label> - Jump when less than or equal to

- Semantics:

These instructions are conditional jumps that are usually based on a cmp operation



```
b1:
    mov EAX, 1
    cmp EAX, 1
    je exitprogram
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram
```



```
b1:
    mov EAX, 2
    cmp EAX, 1
    jg exitprogram
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram
```

Control Flow Instructions

cmp

- Syntax:

cmp <reg>,<reg>

cmp <reg>,<mem>

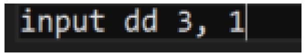
cmp <mem>,<reg>

cmp <reg>,<con>

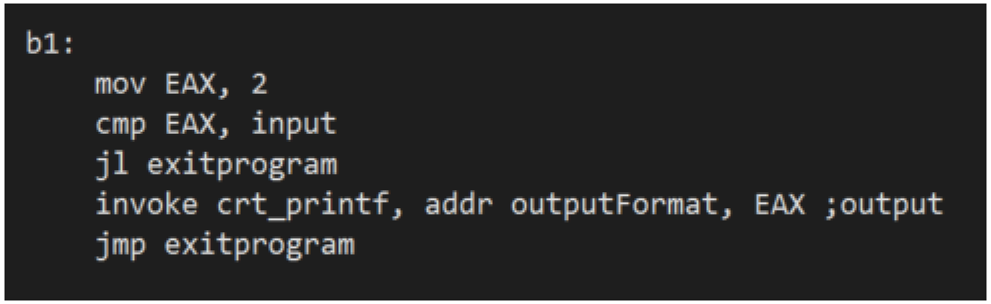
cmp <mem>,<con>

- Semantics:

Compares the data, setting the condition flags in the specific register appropriately.



```
input dd 3, 1
```



```
b1:
    mov EAX, 2
    cmp EAX, input
    jl exitprogram
    invoke crt_printf, addr outputFormat, EAX ;output
    jmp exitprogram
```

Summary

- Data Movement Instructions
- Arithmetic and Logic Instructions
- Control Flow Instructions