Assembly Language Lecture Series:

x86-64 Control Transfer Instructions

Sensei RL Uy, College of Computer Studies,
De La Salle University, Manila, Philippines

Copyright Notice

This lecture contains copyrighted materials and is use solely for instructional purposes only, and not for redistribution.

Do not edit, alter, transform, republish or distribute the contents without obtaining express written permission from the author.

x86-64 Arithmetic Instructions

1. CMP
2. JMP
3. Jcc
4. LOOP
5. JCXZ/JECXZ
6. PUSH
7. POP
8. CALL
9. RET

x86-64 Control Transfer Instructions: CMP

CMP (compare)

Syntax: CMP src1, src2
src1 – src2
result discarded
src1: reg/mem
src2: reg/mem/imm8_16_32

Flags affected:
*all status flags

Note:

1. Immediate value up to **32-bit** only
2. When an **immediate value** is used as an **operand**, it is **sign-extended** to the **length** of the **destination** operand format
3. **Negative** number in **hex** has to be sign-extended to **64-bit**

x86-64 Control Transfer Instructions: CMP

CMP (compare)

Syntax: CMP src1, src2
src1 – src2
result discarded
src1: reg/mem
src2: reg/mem/imm8_16_32

Flags affected:
*all status flags

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0006
MOV RBX, 0x0000_0000_0000_0006
CMP RAX, RBX
```

1. **What will RAX contain after execution?**
2. **What will be the value of the status flags after execution?**

x86-64 Control Transfer Instructions: CMP

CMP (compare)

Syntax: CMP src1, src2
src1 – src2
result discarded
src1: reg/mem
src2: reg/mem/imm8_16_32

Flags affected:
*all status flags

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0006
MOV RBX, 0x0000_0000_0000_0006
CMP RAX, RBX
```

1. What will RAX contain after execution?
2. What will be the value of the status flags after execution?

RAX = 0000000000000006
CF = 0
SF = 0
ZF = 1
OF = 0
PF = 1
AF = 0

For readability:
0000_0000_0000_0006

x86-64 Control Transfer Instructions: JMP

JMP (Jump instruction)

Unconditional branch
JMP <label>

Flags affected: none

x86-64 Control Transfer Instructions: JMP

JMP (Jump instruction)

Unconditional branch
JMP <label>

Flags affected: none

Example:

```
section .text
MOV CL, -2
JMP L1
MOV CL, 1
L1: NOP
```

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JMP

JMP (Jump instruction)

Unconditional branch
JMP <label>

Flags affected: none

Example:

```
section .text
MOV CL, -2
JMP L1
MOV CL, 1
L1: NOP
```

1. What will CL contain after execution?

   CL = FE

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0xFF
CMP AL, BL
JL L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0xFF
CMP AL, BL
JL L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < BL
then goto L1
else next line

1. What will CL contain after execution?

CL = FF

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 56
MOV BL, 45
CMP AL, BL
JL L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 56
MOV BL, 45
CMP AL, BL
JL L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < BL
then goto L1
else next line

1. What will CL contain after execution?

CL = 00

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0x45
MOV BL, 0xFE
CMP AL, BL
JG L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL > BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Signed):
JL: jump if less than
JG: jump if greater than
JE: jump if equal
JNE: jump if not equal
JGE: jump if >=
JLE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0x45
MOV BL, 0xFE
CMP AL, BL
JG L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL > BL
then goto L1
else next line

1. What will CL contain after execution?

CL = FF

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0xFF
CMP AL, BL
JB L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < unsigned BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0xFF
CMP AL, BL
JB L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < unsigned BL
then goto L1
else next line

1. What will CL contain after execution?

CL = FF

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0x56
MOV BL, 0x45
CMP AL, BL
JB L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < unsigned BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0x56
MOV BL, 0x45
CMP AL, BL
JB L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL < unsigned BL
then goto L1
else next line

1. What will CL contain after execution?

CL = 00

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0x45
CMP AL, BL
JA L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL > unsigned BL
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Unsigned):
JB: jump if less than
JA: jump if greater than
JAE: jump if >=
JBE: jump if <=

Flags affected: none

Example:

```
section .text
MOV AL, 0xFE
MOV BL, 0x45
CMP AL, BL
JA L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL > unsigned BL
then goto L1
else next line

1. What will CL contain after execution?

CL = FF

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Using Flag):

JC: jump if CF = 1
JNC: jump if CF = 0
JZ: jump if ZF = 1
JNZ: jump if ZF = 0
JS: jump if SF = 1
JNS: jump if SF = 0
JP: jump if PF = 1
JNP: jump if PF = 0
JO: jump if OF = 1
JNO: jump if OF = 0

Flags affected: none

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Using Flag):

JC: jump if CF = 1
JNC: jump if CF = 0
JZ: jump if ZF = 1
JNZ: jump if ZF = 0
JS: jump if SF = 1
JNS: jump if SF = 0
JP: jump if PF = 1
JNP: jump if PF = 0
JO: jump if OF = 1
JNO: jump if OF = 0

Flags affected: none

Example:

```
section .text
MOV AL, 0xFF
INC AL
JZ L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL == 0
then goto L1
else next line

1. **What will CL contain after execution?**

x86-64 Control Transfer Instructions: JCC

Jcc <label>

Conditional branch (Using Flag):

JC: jump if CF = 1
JNC: jump if CF = 0
JZ: jump if ZF = 1
JNZ: jump if ZF = 0
JS: jump if SF = 1
JNS: jump if SF = 0
JP: jump if PF = 1
JNP: jump if PF = 0
JO: jump if OF = 1
JNO: jump if OF = 0

Flags affected: none

Example:

```
section .text
MOV AL, 0xFF
INC AL
JZ L1
MOV CL, 0x00
JMP FINIS
L1: MOV CL, 0xFF
FINIS: NOP
```

If AL == 0
then goto L1
else next line

1. What will CL contain after execution?

CL = FF

x86-64 Control Transfer Instructions: LOOP

LOOP (Loop according to RCX counter)

DEC RCX
if RCX ≠ 0 then loop to label
else next instruction

*can also be register ECX or CX

Flags affected: none

x86-64 Control Transfer Instructions: LOOP

LOOP (Loop according to RCX counter)

DEC RCX
if RCX ≠ 0 then loop to label
else next instruction
*can also be register ECX or CX

Flags affected: none

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

1. **What will RCX contain after execution?**
2. **What will AL contain after execution?**

x86-64 Control Transfer Instructions: LOOP

LOOP (Loop according to RCX counter)

DEC RCX
if RCX ≠ 0 then loop to label
else next instruction
*can also be register ECX or CX

Flags affected: none

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

1. What will RCX contain after execution?
2. What will AL contain after execution?

RCX = 0000000000000000
AL = 02

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 3

AL = 5

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 3

AL = 4

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 2

AL = 4

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

Is RCX equal to 0?

RCX = 2

AL = 4

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 2

AL = 4

Is RCX equal to 0?

No, then proceed.

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 2

AL = 4

Continue until RCX is equal to 0

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 1

AL = 3

Continue until RCX is equal to 0

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 0

AL = 2

Continue until RCX is equal to 0

x86-64 Control Transfer Instructions: LOOP

Example:

```
section .text
        MOV RCX, 03
        MOV AL,  05
    L1: DEC AL
        LOOP L1
 FINIS: NOP
```

RCX = 0

AL = 2

Stop at NOP

x86-64 Control Transfer Instructions: JRCXZ, JECXZ, JCXZ

JRCXZ

Syntax: JRCXZ <label>
if RCX == 0 then goto label
Flags affected: none

JECXZ

Syntax: JECXZ <label>
if ECX == 0 then goto label
Flags affected: none

JCXZ

Syntax: CXZ <label>
if CX == 0 then goto label
Flags affected: none

x86-64 Control Transfer Instructions: JRCXZ, JECXZ, JCXZ

JRCXZ

Syntax: JRCXZ <label>
if RCX == 0 then goto label
Flags affected: none

JECXZ

Syntax: JECXZ <label>
if ECX == 0 then goto label
Flags affected: none

JCXZ

Syntax: CXZ <label>
if CX == 0 then goto label
Flags affected: none

Example:

```
section .text
        MOV RCX, 0x0000000000000000
        MOV RBX, 0x0000000000000005
        JRCXZ FINIS
L1:     DEC RBX
        LOOP L1
FINIS: NOP
```

1. **What will RBX contain after execution?**

x86-64 Control Transfer Instructions: JRCXZ, JECXZ, JCXZ

JRCXZ

Syntax: JRCXZ <label>
if RCX == 0 then goto label
Flags affected: none

JECXZ

Syntax: JECXZ <label>
if ECX == 0 then goto label
Flags affected: none

JCXZ

Syntax: CXZ <label>
if CX == 0 then goto label
Flags affected: none

Example:

```
section .text
        MOV RCX, 0x0000000000000000
        MOV RBX, 0x0000000000000005
        JRCXZ FINIS
L1:     DEC RBX
        LOOP L1
FINIS: NOP
```

1. What will RBX contain after execution?

RBX = 0000000000000005

Exercise

Write an x86-64 program to count the number of elements in the list found in memory location ALPHA. The list is terminated by 0. Store the count in memory location COUNT. Output the count as well.

```
section .data
ALPHA db 0x12, 0x34, 0x56, 0x99, 0x00
COUNT db 0x00

section .text
global CMAIN
CMAIN:
;write your code here
xor rax, rax
ret
```

address	Memory data (byte)
COUNT	00
	00
	99
	56
	34
ALPHA	12

x86-64 Control Transfer Instructions: PUSH

PUSH(Push onto Stack)

PUSH src
src: r16_64
src: m16_64
src: imm8_16_32
8-bit immediate is sign-extended
{
  RSP ← RSP – 8 |
  RSP ← RSP – 2
}
[RSP] ← source

***No flags affected**

Example:

```
section .text
        MOV RAX, 0x123456789ABCDEF0
        PUSH RAX
```

NOTE: **Does not support 32-bit operands**
immediate is treated as 64-bit sign-extended

x86-64 Control Transfer Instructions: PUSH

PUSH(Push onto Stack)

Example:

```
section .text
        MOV RAX, 0x123456789ABCDEF0
        PUSH RAX
```

address	Stack Memory
0060FE39	
0060FE38	12
0060FE37	34
0060FE36	56
0060FE35	78
0060FE34	9A
0060FE33	BC
0060FE32	DE
0060FE31	F0

x86-64 Control Transfer Instructions: POP

POP (pop a value from the stack)

POP dst
dst: r16_64
dst: m16_64

dst←[rsp]
{
RSP ← RSP + 8 |
RSP ← RSP + 2
}

***No flags affected**

Example:

```
section .text
        POP RAX
```

NOTE: **Does not support 32-bit operands**

x86-64 Control Transfer Instructions: POP

POP (pop a value from the stack)

Example:

```
section .text
        POP RAX
```

address	Stack Memory
0060FE39	
0060FE38	12
0060FE37	34
0060FE36	56
0060FE35	78
0060FE34	9A
0060FE33	BC
0060FE32	DE
0060FE31	F0

RSP

RAX (register)
~~11~~ 12
~~33~~ 34
~~55~~ 56
~~77~~ 78
~~99~~ 9A
~~BB~~ BC
~~DD~~ DE
~~F0~~ F0

x86-64 Control Transfer Instructions: CALL/RET

CALL

The address of the next instruction (RIP) is push to the stack

RET

The 8-byte data is POP'd to RIP

Example:

```
  call Test
       xor eax, eax
       ret
Test:
       PRINT_STRING  "Stack"
       ret
```

STACK MEMORY

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
  call Test
       xor rax, rax
       ret
Test:
       PRINT_STRING  "Stack"
       ret
```

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF18

STACK MEMORY

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
call Test
        xor rax, rax
        ret
Test:
        PRINT_STRING  "Stack"
        ret
```

`call Test` pushes the return address

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF10

STACK MEMORY

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
call Test
     xor rax, rax
     ret
Test:
     PRINT_STRING  "Stack"
     ret
```

Then jumps to `Test` function

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF10

STACK MEMORY

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
  call Test
       xor rax, rax
       ret
Test:
       PRINT_STRING "Stack"
       ret
```

Execute function command

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF10

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
  call Test
       xor rax, rax
       ret
Test:
       PRINT_STRING  "Stack"
       ret
```

`ret` pops the return address and give to RIP

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

RSP

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF10

STACK MEMORY

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
call Test
     xor rax, rax
     ret
Test:
     PRINT_STRING  "Stack"
     ret
```

Then jumps to return address based on RIP

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF18

assume return address
of this 0x0000 0000 0040 14E5

Example:

```
call Test
     xor rax, rax
     ret
Test:
     PRINT_STRING  "Stack"
     ret
```

Lastly, return properly to
0x0000_0000_0040_13A5

STACK MEMORY

RSP

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	00
0061FF16	00
0061FF15	00
0061FF14	00
0061FF13	00
0061FF12	40
0061FF11	14
0061FF10	E5
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0x061FF20

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

RSP

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	
RBX	
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

RSP

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	1111_1111_1111_1111
RBX	
RCX	
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

RSP

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

RSP

Lower memory address

STACK MEMORY

0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	
0061FF0A	
0061FF09	
0061FF08	
0061FF07	
0061FF06	
0061FF05	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	
0061FF06	
0061FF05	
0061FF04	
0061FF03	
0061FF02	
0061FF01	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	
0061FF06	
0061FF05	
0061FF04	
0061FF03	
0061FF02	
0061FF01	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the

Push the return address 0x0000_0000_0040_13A1

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Then jump to the address of func1

RSP

Lower memory address

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

prints 0x0000_0000_0040_13A1

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
F02	40
F01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Higher memory address

Lower memory address

STACK MEMORY

0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	2222_2222_2222_2222

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

prints 0x2222_2222_2222_2222

RSP

Higher memory address

Lower memory address

STACK MEMORY

0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	2222_2222_2222_2222

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

return to the address 0x1111_1111_1111_1111

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	2222_2222_2222_2222

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   pop   rcx
         PRINT_HEX 8, rcx
   pop   rdx
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

THE PROGRAM RETURNS TO A WRONG ADDRESS!

return to the address 0x1111_1111_1111_1111

Higher memory address

Lower memory address

RSP

STACK MEMORY

0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
FF10	11
FF0F	22
0061FF0E	22
0061FF0D	22
F0C	22
F0B	22
F0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0011_2233_4455_6677 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	0000_0000_0040_13A1
RDX	2222_2222_2222_2222

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	
0061FF16	
0061FF15	
0061FF14	
0061FF13	
0061FF12	
0061FF11	
0061FF10	
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Lower memory address

RSP

STACK MEMORY

0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	
0061FF0E	
0061FF0D	
0061FF0C	
0061FF0B	
0061FF0A	
0061FF09	
0061FF08	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	
0061FF06	
0061FF05	
0061FF04	
0061FF03	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Lower memory address

STACK MEMORY

0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	
0061FF06	
0061FF05	
0061FF04	
0061FF03	

RSP

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	
0061FEF6	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RSP

Lower memory address

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	
0061FEF6	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

RBP

RSP

Lower memory address

STACK MEMORY

0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	
0061FEFB	
0061FEFA	
0061FEF9	
0061FEF8	
0061FEF7	
0061FEF6	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	
RDX	
RBP	0000_0000_0061_FF00

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Higher memory address

Lower memory address

RBP

RSP

STACK MEMORY

0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	
RBP	0000_0000_0061_FF00

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_**RBP**

Lower memory address

STACK MEMORY

0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_ **RBP**

Lower memory address

STACK MEMORY

0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00

rcx = 2222_2222_2222_2222
rdx = 1111_1111_1111_1111

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

assume the return address is 0x0000_0000_0040_13A1

Higher memory address

Lower memory address

RBP

RSP

STACK MEMORY

0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF00
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

Higher memory address

RBP

assume the return address is 0x0000_0000_0040_13A1

RSP

return to the address 0x0000_0000_0040_13A1

Lower memory address

STACK MEMORY

0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00
0061FF03	00
0061FF02	40
0061FF01	13
0061FF00	A1
0061FEFF	
0061FEFE	
0061FEFD	
0061FEFC	

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

RSP

Lower memory address

STACK MEMORY

0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp
   mov   rcx
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

return to the address 0x1111_1111_1111_1111

RSP

Lower memory address

STACK MEMORY

0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

What's Wrong Here?

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp
   mov   rcx
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret
```

Lower memory address

return to the address 0x1111_1111_1111_1111

RSP

THE PROGRAM RETURNS TO A WRONG ADDRESS!

STACK MEMORY

0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

Proper Way

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8,
         NEWLINE
         PRINT_HEX 8,
         ret 16
```

Higher memory address

RBP

assume the return address is 0x0000_0000_0040_13A1

RSP

Optional operand added to RSP after restoring the return address to RIP

Lower memory address

STACK MEMORY

0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22
0061FF0A	22
0061FF09	22
0061FF08	22
0061FF07	00
0061FF06	00
0061FF05	00
0061FF04	00

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF08
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

Proper Way

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret 16
```

Lower memory address

RSP

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF10
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

Proper Way

Higher memory address

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret 16
```

RSP

add +16 to bypass parameters

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF18
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

Proper Way

```
section  .text
   mov   rax, 0x1111_1111_1111_1111
         rbx, 0x2222_2222_2222_2222
   push  rax
   push  rbx
   call  func1
   xor   rax, rax
         ret
func1:
   mov   rbp, rsp
   mov   rcx, [rbp+8]
   mov   rdx, [rbp+16]
         PRINT_HEX 8, rcx
         NEWLINE
         PRINT_HEX 8, rdx
         ret 16
```

return to the address 0x0000_0000_0040_13A5

RSP

Higher memory address

RBP

Lower memory address

STACK MEMORY

address	data
0061FF1F	00
0061FF1E	00
0061FF1D	00
0061FF1C	00
0061FF1B	00
0061FF1A	40
0061FF19	13
0061FF18	A5
0061FF17	11
0061FF16	11
0061FF15	11
0061FF14	11
0061FF13	11
0061FF12	11
0061FF11	11
0061FF10	11
0061FF0F	22
0061FF0E	22
0061FF0D	22
0061FF0C	22
0061FF0B	22

0x0000_0000_0040_13A5 is the return address of SASM

RSP contains the address of the stack memory

RSP	0000_0000_0061_FF20
RAX	1111_1111_1111_1111
RBX	2222_2222_2222_2222
RCX	2222_2222_2222_2222
RDX	1111_1111_1111_1111
RBP	0000_0000_0061_FF00

Translating Standard Control Structures

IF STATEMENT

Pseudo-code:

if (condition) then_block;

Assembly code:

1. Code to set EFLAGS
2. J*cc* endif ; select *cc* so that the condition is false
3. ; code for then_block
4. endif

Translating Standard Control Structures

IF STATEMENT (Example)

```
if (RSI==RDI)
  then EDX = 0x12345678;
```

Assembly code:

	CMP RSI, RDI
	JNE endif
	MOV EDX, 0x12345678
endif:	; next instruction

Translating Standard Control Structures

IF-ELSE STATEMENT

Pseudo-code:

if (condition) then_block;
else else_block;

Assembly code:

1. Code to set EFLAGS
2. Jcc else_block ; select *cc* so that the condition is false
3. ; code for then_block
4. JMP endif
5. else_block:
6. endif:

Translating Standard Control Structures

IF-ELSE STATEMENT (Example)

```
if (RSI==RDI) then EDX = 0x12345678;
         else EDX = 0x87654321
```

Assembly code:

	CMP RSI, RDI
	JNE else
	MOV EDX, 0x12345678
	JMP endif
else:	MOV EDX, 0x87654321
endif:	; next instruction

Translating Standard Control Structures

WHILE LOOP STATEMENT

Pseudo-code (Condition Upon Entry):

```
while (condition) {
        body of loop; }
```

Assembly code:

1. while:
2. Code to set EFLAGS
3. Jcc endwhile ; select *cc* so that the condition is false
4. ; body of loop
5. JMP while
6. endwhile:

Translating Standard Control Structures

WHILE LOOP STATEMENT (Example)

```
while (RCX<>0)
  {EDX = EDX+1
   RCX = RCX-1};
```

Assembly code:

while:	**CMP RCX,0000000000000000**
	JE endwhile
	INC EDX
	DEC RCX
	JMP while
endwhile:	; next instruction

Translating Standard Control Structures

DO-WHILE LOOP STATEMENT

Pseudo-code (Condition Upon Exit):

do { body of loop; } while (condition)

Assembly code:

1. do:
2. ; body of loop
3. ; Code to set EFLAGS
4. Jcc do ; select cc so that the condition is true

Translating Standard Control Structures

DO-WHILE LOOP STATEMENT (Example)

```
do
 {EDX = EDX+1;
  ECX = RCX-1;}
while RCX <>0
```

Assembly code:

do:	**INC EDX**
	DEC RCX
	CMP ECX,00000000
	JNE do
	; next instruction

Translating Standard Control Structures

SWITCH STATEMENT

Pseudo-code:

```
SWITCH case_var
        {{label_list}...:stmt; break};
DEFAULT
        statements; break;
END;
```

Assembly code:

	CMP case_var, case_list_1
	JNE label_list_2
	; instructions for case_list_1
	JMP end
label_list_2:	CMP case_var, case_list_2
	JNE label_list_n
	; instructions for case_list_2
	JMP end
label_list_n:	CMP case_var, case_list_other
	JNE default
	; instructions for case_list_n
	JMP end
default:	; instructions for case_list_default
end:	; next instructions
	CMP case_var, case_list_1

Translating Standard Control Structures

SWITCH STATEMENT (Example)

```
SWITCH (RSI) {
        1: EBX = EBX +1; break;
        2: EBX = EBX – 1; break;
        3: EBX = EBX + EBX; break;
        DEFAULT
        EBX = EBX – EBX; break;
}
```

Assembly code:

	CMP RSI, 0000000000000001
	JNE next_case_2
	INC EBX
	JMP end
next_case_2:	CMP ESI, 0000000000000002
	JNE next_case_3
	DEC EBX
	JMP end
next_case_3:	CMP ESI, 0000000000000003
	JNE default
	ADD EBX, EBX
	JMP end
default:	SUB EBX, EBX
end:	; next instructions

Translating Standard Control Structures

FOR-NEXT LOOP STATEMENT

Pseudo-code (Condition Upon Exit):

```
for var = initial_value to final_value do
        {body of loop};
```

Assembly code:

1. Initialized CX/ECX/RCX
2. for:
3. ; body of loop
4. LOOP for

Translating Standard Control Structures

FOR-NEXT LOOP STATEMENT (Example)

```
For ECX = 1 to 5 do EDX = EDX +1
```

Assembly code:

	MOV RCX, 0000000000000005
for:	INC EDX
	LOOP for
	; next instruction