Assembly Language Lecture Series:

x86-64 Arithmetic Instructions

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

Note: Some of the content here are the same from previous materials.

Copyright Notice

64-bit number range (unsigned)

	Largest positive number (hex)	Largest positive (decimal)
8-bit	0xFF	255
16-bit	0xFFFF	65535
32-bit	0xFFFF_FFFF	4294967295
64-bit	0xFFFF_FFFF_FFFF_FFFF	18446744073709551615 (18.x * 10^{18})

64-bit number range (signed)

	Largest positive number (hex)	Largest positive (decimal)	Largest magnitude negative (hex)	Largest magnitude negative (decimal)
8-bit	0x7F	127	0x80	-128
16-bit	0x7FFF	32767	0x8000	-32768
32-bit	0x7FFF_FFFF	2147483647	0x8000_0000	-2147483648

	Largest positive number (hex)	Largest positive (decimal)
64-bit	0x7FFF_FFFF_FFFF_FFFF	9223372036854775807 (9.x * 10^{18})

	Largest magnitude negative (hex)	Largest magnitude negative (decimal)
64-bit	0x8000_0000_0000_0000	-9223372036854775808 (-9.x * 10^{18})

x86-64 Arithmetic Instructions

1. **ADD**
   Addition instruction
2. **INC**
   Increment Instruction
3. **SUB**
   Subtraction Instruction
4. **DEC**
   Decrement Instruction
5. **NEG**
   2’s complement negation
6. **IMUL**
   signed multiplication
7. **MUL**
   unsigned multiplication
8. **IDIV**
   signed division
9. **DIV**
   unsigned division

x86-64 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src
dst ←dst + src
dst: reg/mem
src: 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 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src

dst ←dst + src

dst: reg/mem

src: reg/mem/imm8_16_32

Flags affected:

*all status flags

Example:

```
section .data
var1 dq 0x7FFF_FFFF_FFFF_FFFE
section .text
MOV RAX, 0x01
ADD RAX, [var1]
```

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

x86-64 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src
dst ←dst + src
dst: reg/mem
src: reg/mem/imm8_16_32

Flags affected:
*all status flags

Example:

```
section .data
var1 dq 0x7FFF_FFFF_FFFF_FFFE
section .text
MOV RAX, 0x01
ADD RAX, [var1]
```

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

RAX = 7FFFFFFFFFFFFFFF	OF = 0
CF = 0	PF = 1
SF = 0	AF = 0
ZF = 0	

For readability:
7FFF_FFFF_FFFF_FFFF

x86-64 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src
dst ←dst + src
dst: reg/mem
src: reg/mem/imm8_16_32

Flags affected:
*all status flags

Example: add RAX with -1

- `ADD RAX, -1`
- `ADD RAX, 0xFF`
- `ADD RAX, 0xFFFF`
- `ADD RAX, 0xFFFF_FFFF`
- `ADD RAX, 0xFFFF_FFFF_FFFF_FFFF`

x86-64 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src

dst ←dst + src

dst: reg/mem

src: reg/mem/imm8_16_32

Flags affected:

*all status flags

Example: add RAX with +10

- `ADD RAX, 10`
- `ADD RAX, 0x0A`
- `ADD RAX, 0x000A`
- `ADD RAX, 0x0000_0000A`
- `ADD RAX, 0x0000_0000_0000_000A`

x86-64 Arithmetic Instructions: ADD

ADD (addition instruction)

Syntax: ADD dst, src
dst ←dst + src
dst: reg/mem
src: reg/mem/imm8_16_32

Flags affected:
*all status flags

Example: add RAX with max value

- `ADD RAX, 0x0000_0000_7fff_ffff ; pos`
- `ADD RAX, 2147483647 ; pos`
- `ADD RAX, 0xffff_ffff_8000_0000 ; neg`
- `ADD RAX, -2147483648 ; neg`

x86-64 Arithmetic Instructions: INC

INC (increment instruction)

Syntax: INC dst
dst←dst + 1
dst: reg/mem

Flags affected:
*SF, ZF, OF, PF, AF
CF – not affected

x86-64 Arithmetic Instructions: INC

INC (increment instruction)

Syntax: INC dst

dst←dst + 1

dst: reg/mem

Flags affected:

*SF, ZF, OF, PF, AF

CF – not affected

Example:

```
section .data
var1 dq 0x7FFF_FFFF_FFFF_FFFD
section .text
INC qword[var1]
```

1. **What will memloc var1 contain after execution?**
2. **What will SF, ZF, OF, PF, AF after execution?**

x86-64 Arithmetic Instructions: INC

INC (increment instruction)

Syntax: INC dst
dst←dst + 1
dst: reg/mem

Flags affected:
*SF, ZF, OF, PF, AF
CF – not affected

Example:

```
section .data
var1 dq 0x7FFF_FFFF_FFFF_FFFD
section .text
INC qword[var1]
```

1. What will memloc var1 contain after execution?
2. What will SF, ZF, OF, PF, AF after execution?

var1 = 7FFF_FFFF_FFFF_FFFE
SF = 0
ZF = 0
OF = 0
PF = 0
AF = 0

x86-64 Arithmetic Instructions: **SUB**

SUB (subtraction instruction)

Syntax: SUB dst, src
dst ← dst – src
dst: reg/mem
src: 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 Arithmetic Instructions: SUB

SUB (subtraction instruction)

Syntax: SUB dst, src

dst ←dst – src

dst: reg/mem

src: reg/mem/imm8_16_32

Flags affected:

*all status flags

Example:

```
section .text
mov rax, 0x7fff_ffff_ffff_ffff
sub rax, 0x7fff_ffff
```

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

x86-64 Arithmetic Instructions: SUB

SUB (subtraction instruction)

Syntax: SUB dst, src

dst ←dst – src

dst: reg/mem

src: reg/mem/imm8_16_32

Flags affected:

*all status flags

Example:

```
section .text
mov rax, 0x7fff_ffff_ffff_ffff
sub rax, 0x7fff_ffff
```

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

RAX = 7FFF_FFFF_8000_0000
CF = 0
SF = 0
ZF = 0
OF = 0
PF = 1
AF = 0

x86-64 Arithmetic Instructions: DEC

DEC (decrement instruction)

Syntax: DEC dst
dst ←dst – 1
dst: reg/mem

Flags affected:
*SF, ZF, OF, PF, AF
CF – not affected

x86-64 Arithmetic Instructions: DEC

DEC (decrement instruction)

Syntax: DEC dst
dst ←dst – 1
dst: reg/mem

Flags affected:
*SF, ZF, OF, PF, AF
CF – not affected

Example:

```
section .text
MOV RAX, 0xFF00_0000_0000_0000
DEC AL
```

1. **What will RAX contain after execution?**
2. **What will SF, ZF, OF, PF, AF contain after execution?**

x86-64 Arithmetic Instructions: DEC

DEC (decrement instruction)

Syntax: DEC dst
dst ←dst – 1
dst: reg/mem

Flags affected:
*SF, ZF, OF, PF, AF
CF – not affected

Example:

```
section .text
MOV RAX, 0xFF00_0000_0000_0000
DEC AL
```

1. What will RAX contain after execution?
2. What will SF, ZF, OF, PF, AF contain after execution?

RAX = FF00_0000_0000_00FF
SF = 1
ZF = 0
OF = 0
PF = 1
AF = 1

x86-64 Arithmetic Instructions: NEG

NEG (2's complement negation)

Syntax: NEG dst
dst ← 0 – dst (i.e., 2's complement)
dst: reg/mem

Flags affected: all status flag
CF=0 if operand is 0
OF=1 if operand is 0x80 or 0x8000 or 0x80000000 or 0x8000000000000000

x86-64 Arithmetic Instructions: NEG

NEG (2’s complement negation)

Syntax: NEG dst
dst ← 0 – dst (i.e., 2’s complement)
dst: reg/mem

Flags affected: all status flag
CF=0 if operand is 0
OF=1 if operand is 0x80 or 0x8000 or 0x80000000 or 0x8000000000000000

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0005
NEG RAX
```

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

x86-64 Arithmetic Instructions: NEG

NEG (2's complement negation)

Syntax: NEG dst
dst ← 0 – dst (i.e., 2's complement)
dst: reg/mem

Flags affected: all status flag
CF=0 if operand is 0
OF=1 if operand is 0x80 or 0x8000 or 0x80000000 or 0x8000000000000000

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0005
NEG RAX
```

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

AL = FFFF_FFFF_FFFF_FFFB	SF = 1
CF = 1	PF = 0
OF = 0	AF = 1
ZF = 0	

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) **one-operand*

Syntax: IMUL src

AX ← AL * src8

DX:AX ←AX * src16

EDX:EAX ← EAX * src32

RDX:RAX ← RAX * src64

src: reg/mem

Flags affected:

***CF & OF** (CF=OF=0 if the upper half is the signed extension of the lower half)

***ZF, SF, PF, AF are undefined**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ****one-operand***

Syntax: IMUL src
AX ← AL * src8
DX:AX ←AX * src16
EDX:EAX ← EAX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is the signed extension of the lower half)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0002
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
IMUL RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**
3. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) *one-operand*

Syntax: IMUL src
AX ← AL * src8
DX:AX ←AX * src16
EDX:EAX ← EAX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is the signed extension of the lower half)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0002
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
IMUL RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?
3. What will CF and OF contain after execution?

RAX = 0000_0000_0000_0006
RDX = 0000_0000_0000_0000
CF = OF = 0

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) *one-operand*

Syntax: IMUL src
AX ← AL * src8
DX:AX ←AX * src16
EDX:EAX ← EAX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is the signed extension of the lower half)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x7000_0000_0000_0003
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
IMUL RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**
3. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) *one-operand*

Syntax: IMUL src
AX ← AL * src8
DX:AX ←AX * src16
EDX:EAX ← EAX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is the signed extension of the lower half)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x7000_0000_0000_0003
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
IMUL RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?
3. What will CF and OF contain after execution?

RAX = 5000_0000_0000_0009
RDX = 0000_0000_0000_0001
CF = OF = 1

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ****two-operand***

Syntax: IMUL dst, src

dst ← truncate (dst*src)

***dst** = r16, r32 or r64

***src** = reg/mem/imm8_16_32

Flags affected:

***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)

***ZF, SF, PF, AF are undefined**

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 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ***two-operand***

Syntax: IMUL dst, src
dst ← truncate (dst*src)
***dst** = r16, r32 or r64
***src** = reg/mem/imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RCX, 0x0000_0000_0000_0002
IMUL RCX, 0x7FFF_FFFF
```

1. **What will RCX contain after execution?**
2. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ****two-operand***

Syntax: IMUL dst, src
dst ← truncate (dst*src)
***dst** = r16, r32 or r64
***src** = reg/mem/imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RCX, 0x0000_0000_0000_0002
IMUL RCX, 0x7FFF_FFFF
```

1. What will RCX contain after execution?
2. What will CF and OF contain after execution?

RCX = 0000_0000_FFFF_FFFE
CF = OF = 0

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ****two-operand***

Syntax: IMUL dst, src
dst ← truncate (dst*src)
***dst** = r16, r32 or r64
***src** = reg/mem/imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RCX, 0xFFFF_FFFF_FFFF_FFFE
IMUL RCX, 0xFFFF_FFFF_FFFF_FFFF
```

1. **What will RCX contain after execution?**
2. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ***two-operand***

Syntax: IMUL dst, src
dst ← truncate (dst*src)
***dst** = r16, r32 or r64
***src** = reg/mem/imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RCX, 0xFFFF_FFFF_FFFF_FFFE
IMUL RCX, 0xFFFF_FFFF_FFFF_FFFF
```

1. What will RCX contain after execution?
2. What will CF and OF contain after execution?

RCX = 0000_0000_0000_0002
CF = OF = 0

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication)
three-operand

Syntax: IMUL dst, src1, src2
dst ←truncate (src1*src2)
***dst** = r16, r32 or r64
***src1** = reg/mem
***src2** = imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

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 Arithmetic Instructions: IMUL

IMUL (signed multiplication)
three-operand

Syntax: IMUL dst, src1, src2
dst ←truncate (src1*src2)
***dst** = r16, r32 or r64
***src1** = reg/mem
***src2** = imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .data
var1 dq 0xFFFF_FFFF_FFFF_FFFE
section .text
IMUL RCX, [var1],0xFFFF_FFFF_FFFF_FFFE
```

1. **What will RCX contain after execution?**
2. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication)
three-operand

Syntax: IMUL dst, src1, src2
dst ←truncate (src1*src2)
***dst** = r16, r32 or r64
***src1** = reg/mem
***src2** = imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .data
var1 dq 0xFFFF_FFFF_FFFF_FFFE
section .text
IMUL RCX, [var1],0xFFFF_FFFF_FFFF_FFFE
```

1. What will RCX contain after execution?
2. What will CF and OF contain after execution?

RCX = 0000_0000_0000_0004
CF = OF = 0

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication)
three-operand

Syntax: IMUL dst, src1, src2
dst ←truncate (src1*src2)
***dst** = r16, r32 or r64
***src1** = reg/mem
***src2** = imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x7000_0000_0000_0003
IMUL RCX, RAX,0x0000_0000_0000_0003
```

1. **What will RCX contain after execution?**
2. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: IMUL

IMUL (signed multiplication) ****three-operand***

Syntax: IMUL dst, src1, src2
dst ←truncate (src1*src2)
***dst** = r16, r32 or r64
***src1** = reg/mem
***src2** = imm8_16_32

Flags affected:
***CF & OF** (CF=OF=0 if the intermediate product is the same as the truncated result)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x7000_0000_0000_0003
IMUL RCX, RAX,0x0000_0000_0000_0003
```

1. What will RCX contain after execution?
2. What will CF and OF contain after execution?

RCX = 5000_0000_0000_0009
CF = OF = 1

x86-64 Arithmetic Instructions: MUL

MUL (unsigned multiplication)

Syntax: MUL src
AX ← AL * src8
DX:AX ← AX * src16
EDX:EAX ← AX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is of the result is 0)
***ZF, SF, PF, AF are undefined**

x86-64 Arithmetic Instructions: MUL

MUL (unsigned multiplication)

Syntax: MUL src

AX ← AL * src8

DX:AX ← AX * src16

EDX:EAX ← AX * src32

RDX:RAX ← RAX * src64

src: reg/mem

Flags affected:

***CF & OF** (CF=OF=0 if the upper half is of the result is 0)

***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFFE
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0001
MUL RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**
3. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: MUL

MUL (unsigned multiplication)

Syntax: MUL src
AX ← AL * src8
DX:AX ← AX * src16
EDX:EAX ← AX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is of the result is 0)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFFE
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0001
MUL RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?
3. What will CF and OF contain after execution?

RAX = FFFF_FFFF_FFFF_FFFC
RDX = 0000_0000_0000_0001
CF = OF = 1

x86-64 Arithmetic Instructions: MUL

MUL (unsigned multiplication)

Syntax: MUL src
AX ← AL * src8
DX:AX ← AX * src16
EDX:EAX ← AX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is of the result is 0)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0002
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
MUL RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**
3. **What will CF and OF contain after execution?**

x86-64 Arithmetic Instructions: MUL

MUL (unsigned multiplication)

Syntax: MUL src
AX ← AL * src8
DX:AX ← AX * src16
EDX:EAX ← AX * src32
RDX:RAX ← RAX * src64
src: reg/mem

Flags affected:
***CF & OF** (CF=OF=0 if the upper half is of the result is 0)
***ZF, SF, PF, AF are undefined**

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0002
MOV RCX, 0x0000_0000_0000_0003
MOV RDX, 0x0000_0000_0000_0001
MUL RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?
3. What will CF and OF contain after execution?

RAX = 0000_0000_0000_0006
RDX = 0000_0000_0000_0000
CF = OF = 0

x86-64 Arithmetic Instructions: IDIV

IDIV (signed division)

Syntax: IDIV src

AL ←AX div src8

AH ← AX mod src8

AX ← DX:AX div src16

DX ← DX:AX mod src16

EAX ← EDX:EAX div src32

EDX ← EDX:EAX mod src32

RAX ← RDX:RAX div src64

RDX ← RDX:RAX mod src64

src: reg/mem

Flags affected:

*all flags are undefined

x86-64 Arithmetic Instructions: IDIV

IDIV (signed division)

Syntax: IDIV src

AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0009
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0000
IDIV RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**

x86-64 Arithmetic Instructions: IDIV

IDIV (signed division)

Syntax: IDIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0009
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0000
IDIV RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?

RAX = 0000_0000_0000_0004
RDX = 0000_0000_0000_0001

x86-64 Arithmetic Instructions: IDIV

IDIV (signed division)

Syntax: IDIV src

AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFF9
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0xFFFF_FFFF_FFFF_FFFF
IDIV RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**

x86-64 Arithmetic Instructions: IDIV

IDIV (signed division)

Syntax: IDIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFF9
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0xFFFF_FFFF_FFFF_FFFF
IDIV RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?

RAX = FFFF_FFFF_FFFF_FFFD
RDX = FFFF_FFFF_FFFF_FFFF

x86-64 Arithmetic Instructions: DIV

DIV (unsigned division)

Syntax: DIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Note:

1. **Arithmetic exception** (divide error) will occur if
   a. Divide by 0
   b. The resulting quotient is too large to fit in the destination register

x86-64 Arithmetic Instructions: DIV

DIV (unsigned division)

Syntax: DIV src
AL ←AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0009
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0000
DIV RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**

x86-64 Arithmetic Instructions: DIV

DIV (unsigned division)

Syntax: DIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0x0000_0000_0000_0009
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0x0000_0000_0000_0000
DIV RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?

RAX = 0000_0000_0000_0004
RDX = 0000_0000_0000_0001

x86-64 Arithmetic Instructions: DIV

DIV (unsigned division)

Syntax: DIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFF9
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0xFFFF_FFFF_FFFF_FFFF
IDIV RCX
```

1. **What will RAX contain after execution?**
2. **What will RDX contain after execution?**

x86-64 Arithmetic Instructions: DIV

DIV (unsigned division)

Syntax: DIV src
AL ← AX div src8
AH ← AX mod src8
AX ← DX:AX div src16
DX ← DX:AX mod src16
EAX ← EDX:EAX div src32
EDX ← EDX:EAX mod src32
RAX ← RDX:RAX div src64
RDX ← RDX:RAX mod src64
src: reg/mem

Flags affected:
*all flags are undefined

Example:

```
section .text
MOV RAX, 0xFFFF_FFFF_FFFF_FFF9
MOV RCX, 0x0000_0000_0000_0002
MOV RDX, 0xFFFF_FFFF_FFFF_FFFF
IDIV RCX
```

1. What will RAX contain after execution?
2. What will RDX contain after execution?

Arithmetic exception, divide error