COMP 520 - Compilers

Lecture 14 – PA4 Intro

PA3 Reminder

- It is due this Friday at 11:59pm
- If you haven't started, start immediately!

- Can be somewhat tricky when you reach QualRef

Recall

- These assignments build on each other.
- By thinking ahead and preparing for PA4, you know the goal of PA3

Quick Review of PA3

Not just how to do it, but important notes related to PA4

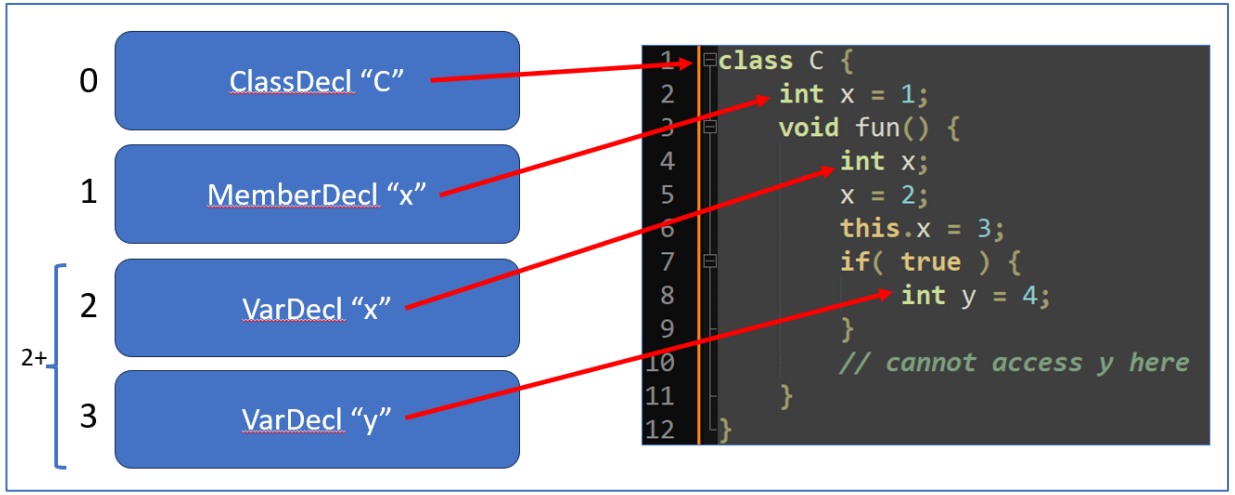
Identification

- Idea: Map each identifier to a Declaration
- A declaration is:
  - Vardecl
  - ParameterDecl

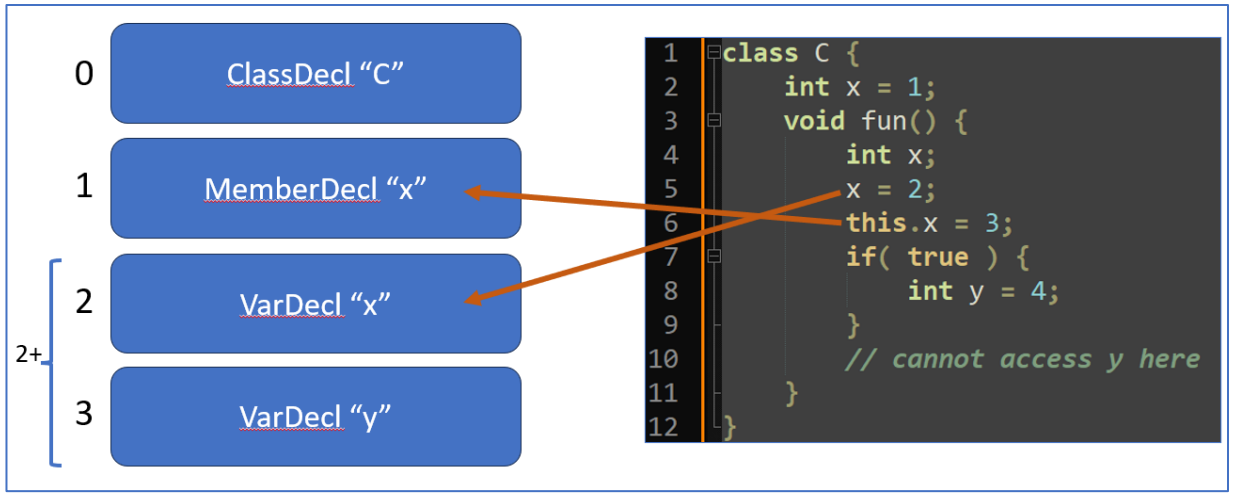
  } LocalDecl
  - FieldDecl
  - MethodDecl

  } MemberDecl

Identification



Red Arrows:
IDTable goes from "String" to Declaration

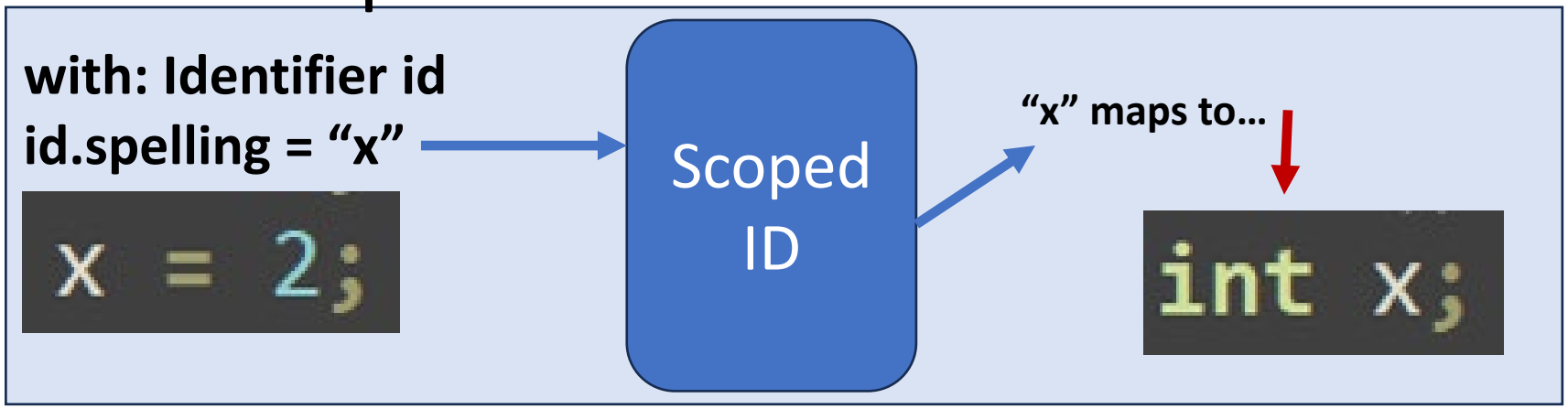


Orange Arrows:

Identifier checks IDTable for "where am I declared?"

Identification Goal

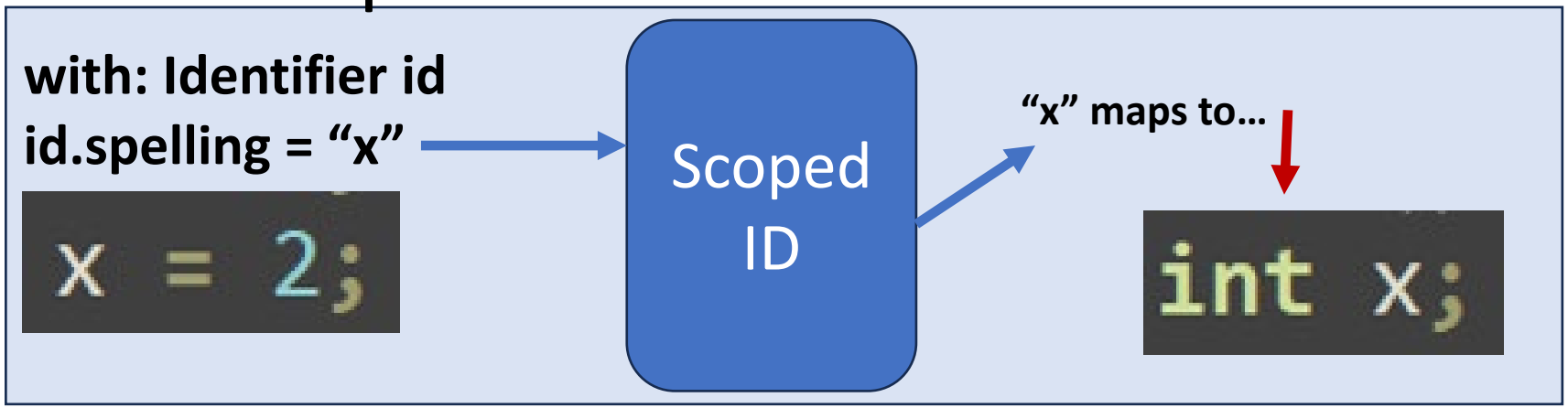
1. Look up where the identifier is declared.



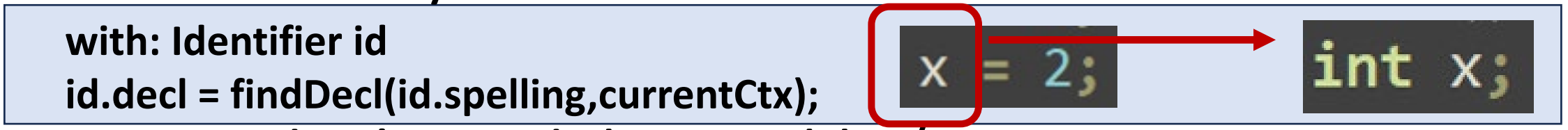
with: Identifier id
id.decl = findDecl(id.spelling,currentCtx);

Identification Goal

1. Look up where the identifier is declared.



2. Store it in my Identifier AST's field "decl"



3. Now I don't need the IDTable / SI

Identification

- All references, except `ThisRef`, contain an identifier.
- This means if every `Identifier` maps to a `Declaration`…
- If you take special care for "`ThisRef`," then all references also map to an `Identifier`, which maps to a `Declaration`.

QualRef Quick Review

From Lec09

QualRef Definition

- Two Fields:
  - Reference ref;
  - Identifier id;

- Grammar: (this|id)(.id)*
- The grammar is misleading.
  The grammar is meant for parsing.

In this example..

- Class A: contains member: B b;
- Class B: contains member: C c;
- Class C: contains member: int x;

Where is a.b.c.x?

```
class A {
    B b;
    int x;
}

class B {
    C c;
    int x;
}

class C {
    int x = 2;
    void fun() {
        int b = 3;
        int c = 4;
        int x = 5;

        A a = new A();
        a.b.c.x = 6;
    }
}
```

Consider: a.b.c.x

QualRef: a.b.c.x

Reference ref = “a.b.c”

Identifier id = “x”

Same as slide 81, Lec09

Consider: a.b.c.x

Reference ref = “a.b.c”

Identifier id = “x”

Strategy: Visit LHS, resolve RHS in context of LHS

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x
a.b.c	a.b (QualRef)	c
a.b	a (IdRef)	b

VISIT

VISIT

VISIT

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x
a.b.c	a.b (QualRef)	c
a.b	a (IdRef) VISIT	b
Context of “a” is class “A”, resolve “b” in ctx of “A”		

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x
a.b.c	a.b (QualRef)	c
a.b	CTX: "A" VISIT	b
Context of "a" is class "A", resolve "b" in ctx of "A"		

RETURN CONTEXT

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x
a.b.c	a.b (QualRef)	c
a.b	CTX: “A”	b Set Decl
Resolve “b” in ctx of “A”		

RETURN CONTEXT

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	a.b.c (QualRef)	x
a.b.c	CTX: “B”	c
a.b	CTX: “A”	b
Resolve “c” in context of class “B”		

Set Decl

RETURN CONTEXT

Consider: a.b.c.x

Strategy: Visit LHS, resolve RHS in context of LHS

QualRef	Left-Hand-Side (ref)	Right-Hand-Side (id)
a.b.c.x	CTX: “C”	x
a.b.c	CTX: “B”	c
a.b	CTX: “A”	b
Resolve “x” in context of class “C”		

Set ID’s Declaration

RETURN CONTEXT

Special Considerations

- Take care for private variable access
- Plan first, code afterwards

Type-Checking

- Covered extensively in earlier lectures
- Recall: no type-casting in miniJava
- Recall: no automatic type-casting in miniJava
- Recall: declarations have a type associated with them. Afterall, this is where they are declared. Thus: if you know an identifier's *declaration*, you know what *type* it is.

Type-Checking (2)

- Your goal is to implement type-checking, not *only* suggestions in the assignment instructions

- At each visit method, ask yourself, “What am I assuming about the types that I have?”

PA4 – Code Generation

What makes a compiler, a compiler!

About a month to do PA4

- PA4 is 3 programming assignments:

1. Assembler (opcode generation)
2. Code generation (organizing your assembly)
3. ELF generation (packaging your code in a way that it can be loaded by x86_64 linux)

Dealing with Decision Paralysis

- If your compiler can generate Linux executable binaries, and the output of that binary matches the autograder, then you get points.

Dealing with Decision Paralysis

- If your compiler can generate Linux executable binaries, and the output of that binary matches the autograder, then you get points.

- This means you have a LOT of control over how you organize your compiler.
- (Note, cannot use others code, or GitHub, etc., etc.)

Optimization

- Your x86_64 code generation does not have to be optimized.
- The thing you're after: get it working
- You can add optimizations in PA5

Optimization – Don't bother yet

- Your x86_64 code generation does not have to be optimized.
- The thing you're after: get it working
- You can add optimizations in PA5

- Plan what you NEED, and FOCUS on those items. The documentation covers ***everything***, but you don't need ***everything*** that x86 has to offer.

Useful Resources

1. ELF File Format: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
2. Online ELF Viewer: http://www.sunshine2k.de/coding/javascript/onlineelfviewer/onlineelfviewer.html
3. Instruction Reference by opcode: http://ref.x86asm.net/coder64.html
4. … sorted by name: http://ref.x86asm.net/coder64-abc.html
5. Instruction Descriptions: https://www.felixcloutier.com/x86/
6. Intel x86_64 documentation: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
7. Decoder/Encoder: https://defuse.ca/online-x86-assembler.htm#disassembly2 (make sure to check x64)

Testing PA4

- When PA4 assignment instructions release, additional testing instructions will release.
- You will be able to remote into a machine where you can use `readelf` and `objdump` binaries.
- You can upload your binaries and INPUT java files, but not your project files.
- Handy for testing.

readelf / objdump

- readelf: Helpful for making sure your ELF file can be read properly
- objdump: Helpful for making sure your encoded assembly is accurate

Let’s get started on x86_64

Target Architecture: x86_64

- The slides and instructions will use x64
- You can still do x86 binaries if you find it easier.
- Target is **LITTLE ENDIAN** (What is Java?)

- Lecture Goals: Target the more difficult x64

Suggested order

1. First do the assembler (convert "`pop rax`" to bytes)
   - Use resources #3, #4, #5, #6
2. Confirm the assembler is working fine (using #7)
3. Do the code generator (visitor implementation)
4. Confirm it is outputting the correct code (using #7)
5. Create ELF scaffolding (using #1, #2)

Suggested order

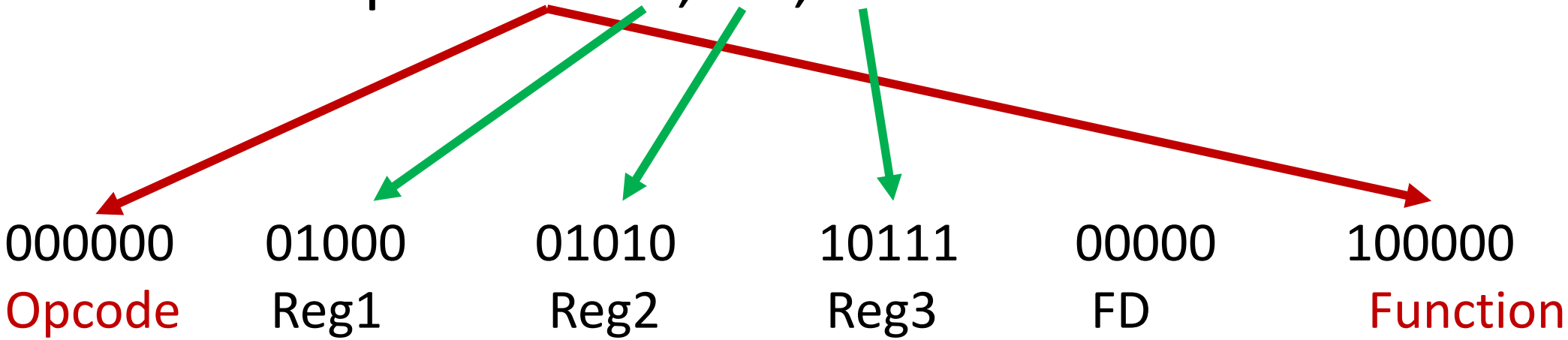
1. First do the assembler (convert "`pop rax`" to bytes)
   - Use resources #3, #4, #5, #6
2. Confirm the assembler is working fine (using #7)
3. Do the code generator (visitor implementation)
4. Confirm it is outputting the correct code (using #7)
5. Create ELF scaffolding (using #1, #2)

Code generation in RISC

- Often: fixed-length instructions
- Often: easy to understand, simple instructions
- MIPS example: add r8,r11,r23

Code generation in RISC

- Often: fixed-length instructions
- Often: easy to understand, simple instructions
- MIPS example: add r8,r11,r23



- MIPS (and other RISC ISAs) is beautiful
- Fixed length operations
- Intuitive to encode and decode

RISC and CISC can't be that different??

- ***Surely***, most things should translate.

- Have an "opcode"
- Have register operands encoded by their index
- Have a "function" if the opcode is shared by several instructions (add, sub, xor, or, cmp, etc.)

Our target architecture is CISC

- We get to use instructions like:

mov dword[**rsi**+**rcx***4+**0x0520**],**edx**

Encoded as:

89 94 8E 20 05 00 00

In-class Exercise, Let's encode!

- I want to encode: **mov rcx,rdx**
- Pseudocode: **rcx := rdx;**

mov rcx,rdx

- Let's say the opcode for this mov is **0x89**
- We find **0x89**, and it says: `mov rm, r`
- So register1 is called "rm", and register2 is called "r"
  - …for now

mov rcx,rdx

- Let's say the opcode for this mov is 0x89
- So far so good, write "**0x89**" to our bytecode

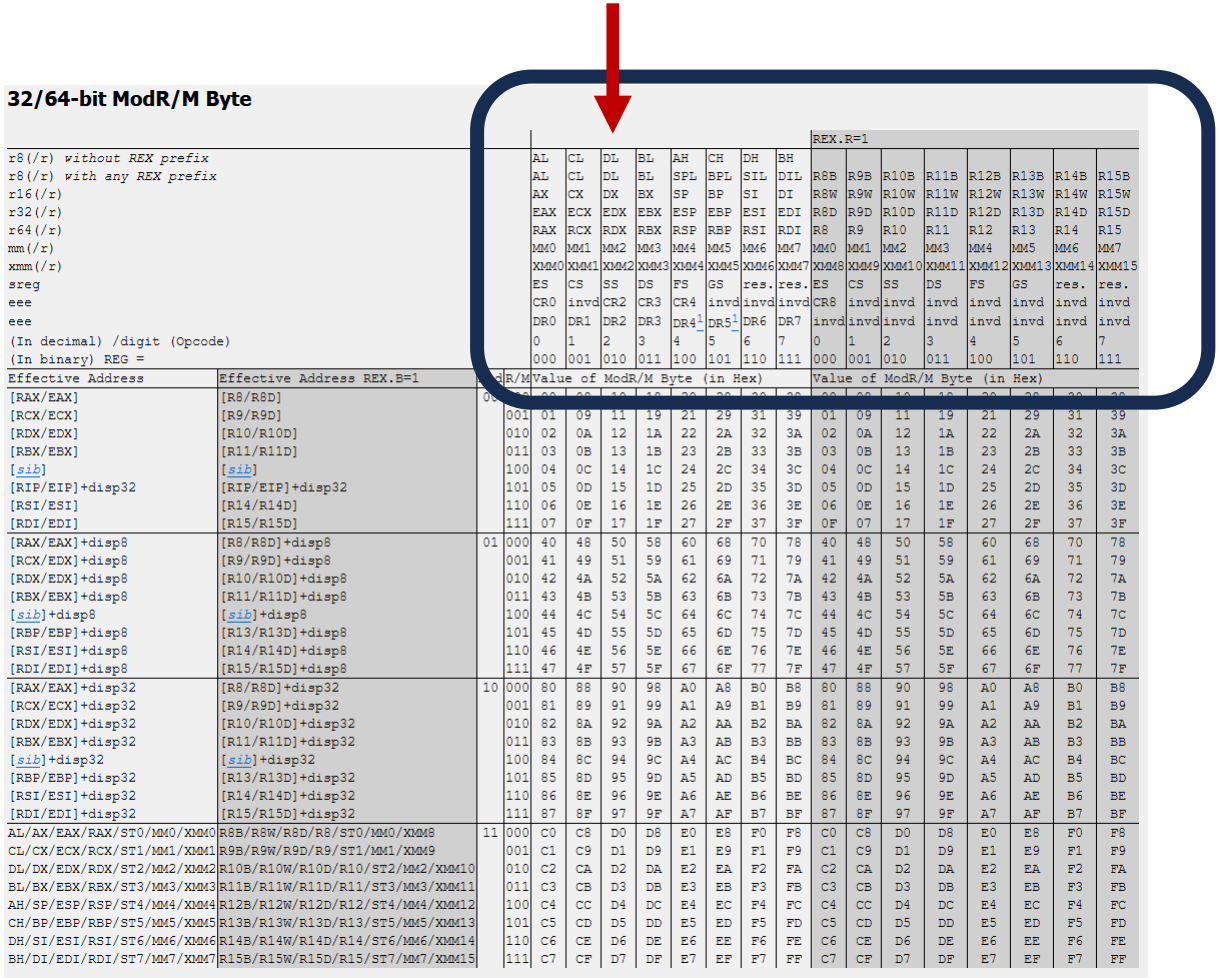
- Go to the register encoding table: http://ref.x86asm.net/coder64-abc.html#modrm_byte_32_64

How do we read this?

32/64-bit ModR/M Byte

												REX.R=1							
r8(/r) *without REX prefix*			AL	CL	DL	BL	AH	CH	DH	BH									
r8(/r) *with any REX prefix*			AL	CL	DL	BL	SPL	BPL	SIL	DIL	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B	
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W	
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D	
r64(/r)			RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8	R9	R10	R11	R12	R13	R14	R15	
mm(/r)			MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	
xmm(/r)			XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15	
sreg			ES	CS	SS	DS	FS	GS	res.	res.	ES	CS	SS	DS	FS	GS	res.	res.	
eee			CR0	invd	CR2	CR3	CR4	invd	invd	invd	CR8	invd	invd	invd	invd	invd	invd	invd	
eee			DR0	DR1	DR2	DR3	DR4[1]	DR5[1]	DR6	DR7	invd	invd	invd	invd	invd	invd	invd	invd	
(In decimal) /digit (Opcode)			0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
(In binary) REG =			000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111	
Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[*sib*]	[*sib*]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[*sib*]+disp8	[*sib*]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[*sib*]+disp32	[*sib*]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

How do we read this? (mov rcx,rdx)



32/64-bit ModR/M Byte

										REX.R=1							
r8(/r) *without REX prefix*		AL	CL	DL	BL	AH	CH	DH	BH								
r8(/r) *with any REX prefix*		AL	CL	DL	BL	SPL	BPL	SIL	DIL	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B
r16(/r)		AX	CX	DX	BX	SP	BP	SI	DI	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
r32(/r)		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
r64(/r)		RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8	R9	R10	R11	R12	R13	R14	R15
mm(/r)		MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
xmm(/r)		XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15
sreg		ES	CS	SS	DS	FS	GS	res.	res.	ES	CS	SS	DS	FS	GS	res.	res.
eee		CR0	invd	CR2	CR3	CR4	invd	invd	invd	CR8	invd	invd	invd	invd	invd	invd	invd
eee		DR0	DR1	DR2	DR3	DR4[1]	DR5[1]	DR6	DR7	invd	invd	invd	invd	invd	invd	invd	invd
(In decimal) /digit (Opcode)		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) REG =		000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)					
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[sib]	[sib]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[sib]+disp8	[sib]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[sib]+disp32	[sib]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Operand “**r**”

Find: RDX

How do we read this? (mov rcx,rdx)

Operand “r”
Find: RDX

Plain registers
Operand “**rm**”
Find: RCX

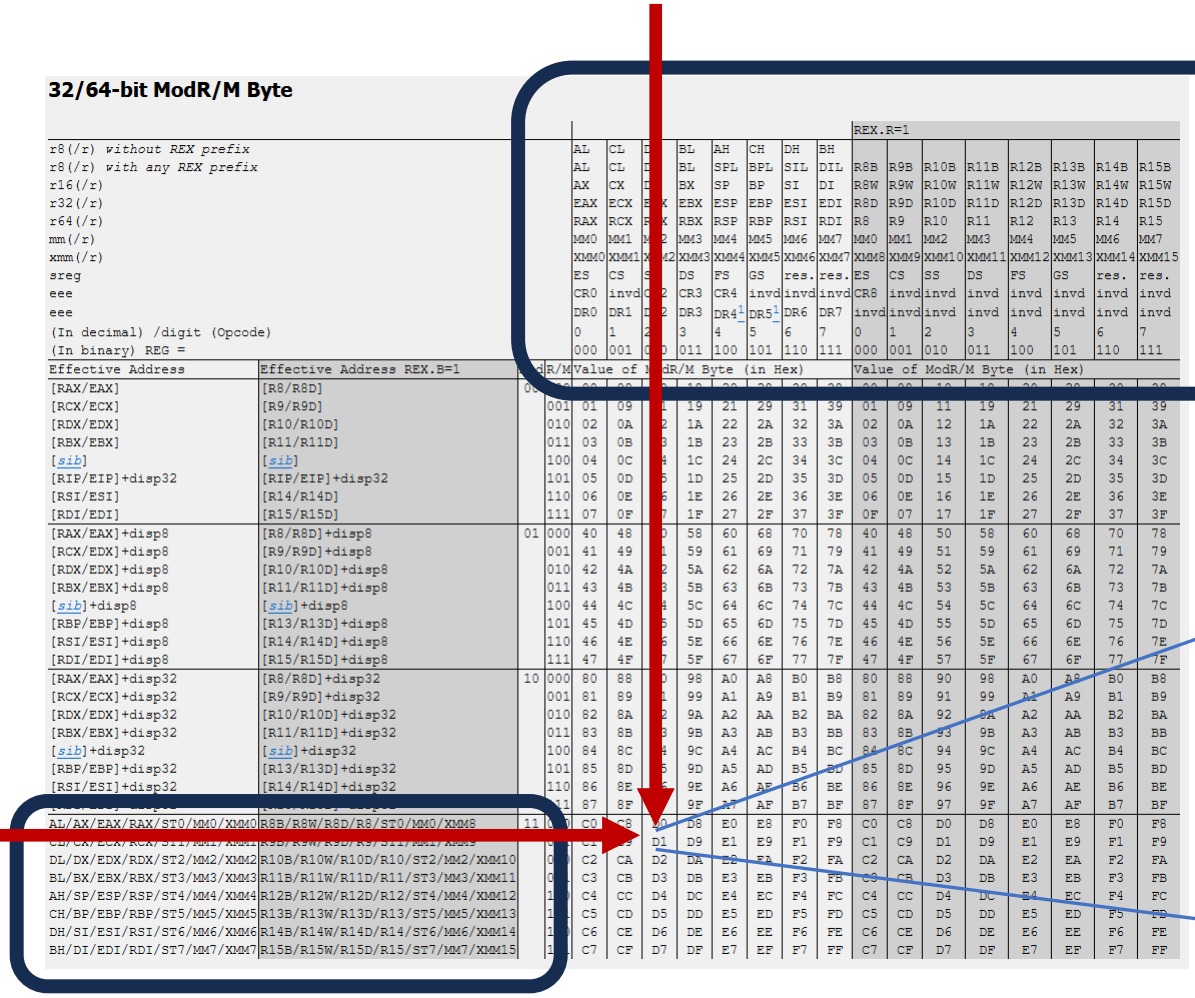
32/64-bit ModR/M Byte

										REX.R=1							
r8(/r) *without REX prefix*		AL	CL	DL	BL	AH	CH	DH	BH								
r8(/r) *with any REX prefix*		AL	CL	DL	BL	SPL	BPL	SIL	DIL	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B
r16(/r)		AX	CX	DX	BX	SP	BP	SI	DI	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
r32(/r)		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
r64(/r)		RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8	R9	R10	R11	R12	R13	R14	R15
mm(/r)		MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
xmm(/r)		XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15
sreg		ES	CS	SS	DS	FS	GS	res.	res.	ES	CS	SS	DS	FS	GS	res.	res.
eee		CR0	invd	CR2	CR3	CR4	invd	invd	invd	CR8	invd	invd	invd	invd	invd	invd	invd
eee		DR0	DR1	DR2	DR3	DR4[1]	DR5[1]	DR6	DR7	invd	invd	invd	invd	invd	invd	invd	invd
(In decimal) /digit (Opcode)		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) REG =		000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)					
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[*sib*]	[*sib*]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[*sib*]+disp8	[*sib*]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[*sib*]+disp32	[*sib*]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

How do we read this? (mov rcx,rdx)

Operand “rm”

Plain registers:
Find: RCX



32/64-bit ModR/M Byte

										REX.R=1							
r8(/r) *without REX prefix*		AL	CL	DL	BL	AH	CH	DH	BH								
r8(/r) *with any REX prefix*		AL	CL	DL	BL	SPL	BPL	SIL	DIL	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B
r16(/r)		AX	CX	DX	BX	SP	BP	SI	DI	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
r32(/r)		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
r64(/r)		RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8	R9	R10	R11	R12	R13	R14	R15
mm(/r)		MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
xmm(/r)		XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15
sreg		ES	CS	SS	DS	FS	GS	res.	res.	ES	CS	SS	DS	FS	GS	res.	res.
eee		CR0	invd	CR2	CR3	CR4	invd	invd	invd	CR8	invd	invd	invd	invd	invd	invd	invd
eee		DR0	DR1	DR2	DR3	DR4[1]	DR5[1]	DR6	DR7	invd	invd	invd	invd	invd	invd	invd	invd
(In decimal) /digit (Opcode)		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) REG =		000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)					
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[*sib*]	[*sib*]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[*sib*]+disp8	[*sib*]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[*sib*]+disp32	[*sib*]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Operand “r”
Find: RDX

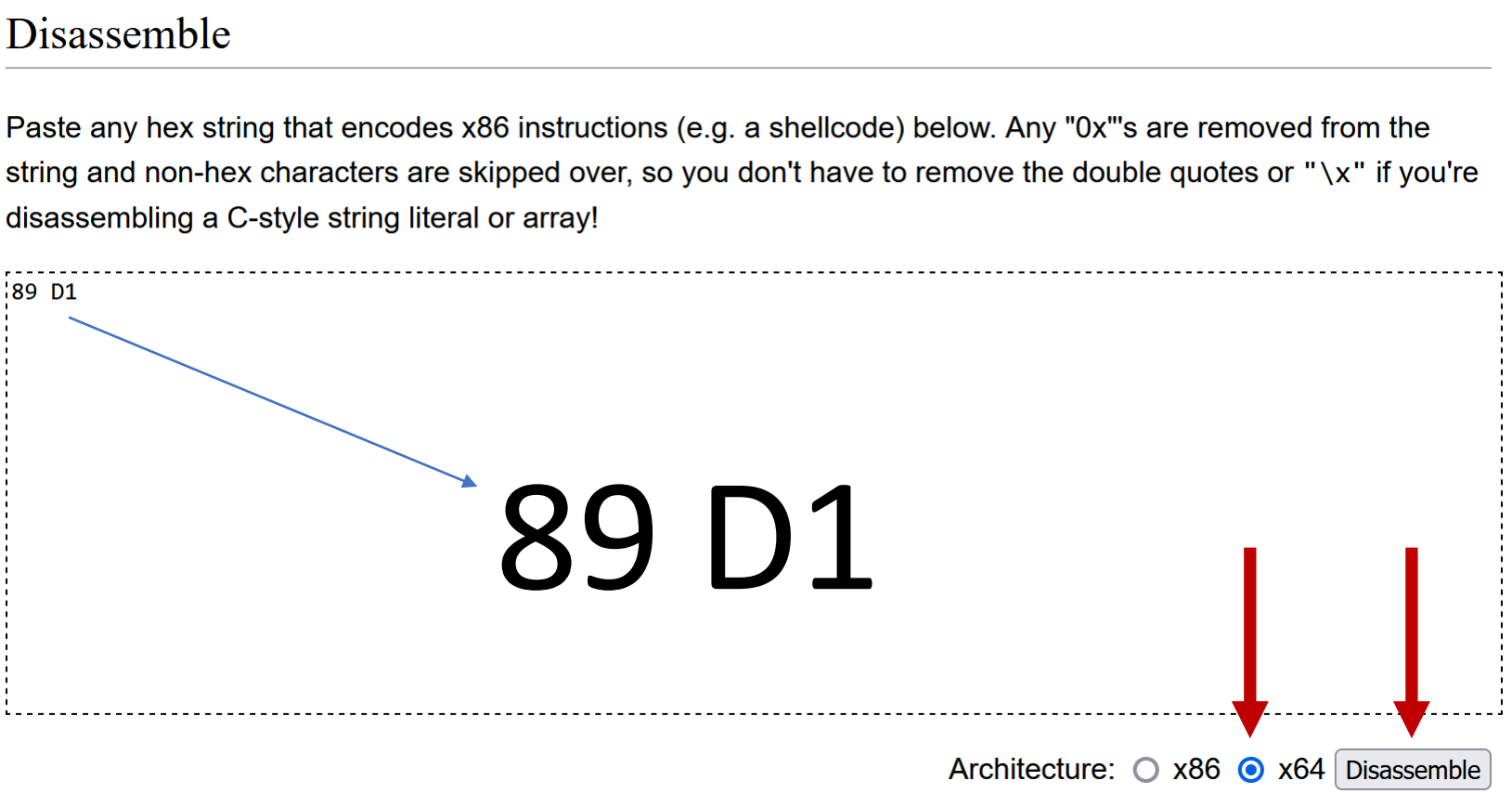
D1

mov rcx,rdx

- So far:
- Opcode= 0x89
- Registers rcx,rdx: 0xD1

- Let's go to our handy-dandy tool: and plug in 89 D1
- https://defuse.ca/online-x86-assembler.htm#disassembly2

Was the result: mov rcx,rdx?



Output was…

mov ecx,edx

We were very close

- If we want to use 64-bit registers, we need to use something called the "**REX**" prefix.

- x86_64 requires you to use "prefix" bytes to determine whether you're using 32-bit registers or 64-bit registers and operands

Recall: New 64-bit registers

- Register Index from [0,7]

  RAX, RCX, RDX, RBX, RSP, RBP, RSI, RDI

- New registers use the same index [0,7]

  R8, R9, R10, R11, R12, R13, R14, R15

REX Prefix: [regB+regX*8+520],regW

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend operand (imm, or register “r”) to 64-bit

Thus, prefix bytes control registers

- Which register gets used is done in the prefix.
- If no prefix is specified, x86_64 will assume **32-bit** for the most part

- Let's try again with: Rex.W (0x48)
- 48 89 D1

Try again!

- Let's try again with: Rex.W (0x48)
- 48 89 D1

https://defuse.ca/online-x86-assembler.htm#disassembly2

Proper output: mov rcx,rdx

What about the rest of that table?

32/64-bit ModR/M Byte

											REX.R=1							
r8(/r) *without REX prefix*		AL	CL	DL	BL	AH	CH	DH	BH									
r8(/r) *with any REX prefix*		AL	CL	DL	BL	SPL	BPL	SIL	DIL	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B	
r16(/r)		AX	CX	DX	BX	SP	BP	SI	DI	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W	
r32(/r)		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D	
r64(/r)		RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8	R9	R10	R11	R12	R13	R14	R15	
mm(/r)		MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7	
xmm(/r)		XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15	
sreg		ES	CS	SS	DS	FS	GS	res.	res.	ES	CS	SS	DS	FS	GS	res.	res.	
eee		CR0	invd	CR2	CR3	CR4	invd	invd	invd	CR8	invd	invd	invd	invd	invd	invd	invd	
eee		DR0	DR1	DR2	DR3	DR4[1]	DR5[1]	DR6	DR7	invd	invd	invd	invd	invd	invd	invd	invd	
(In Binary) REG =		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111	
Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)						
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[*sib*]	[*sib*]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[*sib*]+disp8	[*sib*]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[*sib*]+disp32	[*sib*]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Encoding: mov [rsi+rcx*4+0x520],rdx

- First: find RDX in the column like before

Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[sib]	[sib]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[sib]+disp8	[sib]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[sib]+disp32	[sib]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Encoding: mov [rsi+rcx*4+0x520],rdx

- Ask yourself: is our "displacement of 0x520" zero? One byte? Four bytes?

Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[sib]	[sib]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[sib]+disp8	[sib]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[sib]+disp32	[sib]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Encoding: mov [rsi+rcx*4+0x520],rdx

- Ask yourself: is our "displacement of **0x520**" zero? One byte? **Four bytes**?

Zero

1 byte

4 bytes

Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[*sib*]	[*sib*]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[*sib*]+disp8	[*sib*]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[*sib*]+disp32	[*sib*]+disp32		100	84	8C	94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

Encoding: mov [rsi+rcx*4+0x520],rdx

- Because we have a "**scaled index**" of **rcx*4**, and a displacement of "rsi+0x520", we pick **[SIB]+disp32**

Zero

1 byte

4 bytes

Effective Address	Effective Address REX.B=1	Mod	R/M	Value of ModR/M Byte (in Hex)								Value of ModR/M Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	08	10	18	20	28	30	38	00	08	10	18	20	28	30	38
[RCX/ECX]	[R9/R9D]		001	01	09	11	19	21	29	31	39	01	09	11	19	21	29	31	39
[RDX/EDX]	[R10/R10D]		010	02	0A	12	1A	22	2A	32	3A	02	0A	12	1A	22	2A	32	3A
[RBX/EBX]	[R11/R11D]		011	03	0B	13	1B	23	2B	33	3B	03	0B	13	1B	23	2B	33	3B
[sib]	[sib]		100	04	0C	14	1C	24	2C	34	3C	04	0C	14	1C	24	2C	34	3C
[RIP/EIP]+disp32	[RIP/EIP]+disp32		101	05	0D	15	1D	25	2D	35	3D	05	0D	15	1D	25	2D	35	3D
[RSI/ESI]	[R14/R14D]		110	06	0E	16	1E	26	2E	36	3E	06	0E	16	1E	26	2E	36	3E
[RDI/EDI]	[R15/R15D]		111	07	0F	17	1F	27	2F	37	3F	0F	07	17	1F	27	2F	37	3F
[RAX/EAX]+disp8	[R8/R8D]+disp8	01	000	40	48	50	58	60	68	70	78	40	48	50	58	60	68	70	78
[RCX/EDX]+disp8	[R9/R9D]+disp8		001	41	49	51	59	61	69	71	79	41	49	51	59	61	69	71	79
[RDX/EDX]+disp8	[R10/R10D]+disp8		010	42	4A	52	5A	62	6A	72	7A	42	4A	52	5A	62	6A	72	7A
[RBX/EBX]+disp8	[R11/R11D]+disp8		011	43	4B	53	5B	63	6B	73	7B	43	4B	53	5B	63	6B	73	7B
[sib]+disp8	[sib]+disp8		100	44	4C	54	5C	64	6C	74	7C	44	4C	54	5C	64	6C	74	7C
[RBP/EBP]+disp8	[R13/R13D]+disp8		101	45	4D	55	5D	65	6D	75	7D	45	4D	55	5D	65	6D	75	7D
[RSI/ESI]+disp8	[R14/R14D]+disp8		110	46	4E	56	5E	66	6E	76	7E	46	4E	56	5E	66	6E	76	7E
[RDI/EDI]+disp8	[R15/R15D]+disp8		111	47	4F	57	5F	67	6F	77	7F	47	4F	57	5F	67	6F	77	7F
[RAX/EAX]+disp32	[R8/R8D]+disp32	10	000	80	88	90	98	A0	A8	B0	B8	80	88	90	98	A0	A8	B0	B8
[RCX/ECX]+disp32	[R9/R9D]+disp32		001	81	89	91	99	A1	A9	B1	B9	81	89	91	99	A1	A9	B1	B9
[RDX/EDX]+disp32	[R10/R10D]+disp32		010	82	8A	92	9A	A2	AA	B2	BA	82	8A	92	9A	A2	AA	B2	BA
[RBX/EBX]+disp32	[R11/R11D]+disp32		011	83	8B	93	9B	A3	AB	B3	BB	83	8B	93	9B	A3	AB	B3	BB
[sib]+disp32							94	9C	A4	AC	B4	BC	84	8C	94	9C	A4	AC	B4	BC
[RBP/EBP]+disp32	[R13/R13D]+disp32		101	85	8D	95	9D	A5	AD	B5	BD	85	8D	95	9D	A5	AD	B5	BD
[RSI/ESI]+disp32	[R14/R14D]+disp32		110	86	8E	96	9E	A6	AE	B6	BE	86	8E	96	9E	A6	AE	B6	BE
[RDI/EDI]+disp32	[R15/R15D]+disp32		111	87	8F	97	9F	A7	AF	B7	BF	87	8F	97	9F	A7	AF	B7	BF
AL/AX/EAX/RAX/ST0/MM0/XMM0	R8B/R8W/R8D/R8/ST0/MM0/XMM8	11	000	C0	C8	D0	D8	E0	E8	F0	F8	C0	C8	D0	D8	E0	E8	F0	F8
CL/CX/ECX/RCX/ST1/MM1/XMM1	R9B/R9W/R9D/R9/ST1/MM1/XMM9		001	C1	C9	D1	D9	E1	E9	F1	F9	C1	C9	D1	D9	E1	E9	F1	F9
DL/DX/EDX/RDX/ST2/MM2/XMM2	R10B/R10W/R10D/R10/ST2/MM2/XMM10		010	C2	CA	D2	DA	E2	EA	F2	FA	C2	CA	D2	DA	E2	EA	F2	FA
BL/BX/EBX/RBX/ST3/MM3/XMM3	R11B/R11W/R11D/R11/ST3/MM3/XMM11		011	C3	CB	D3	DB	E3	EB	F3	FB	C3	CB	D3	DB	E3	EB	F3	FB
AH/SP/ESP/RSP/ST4/MM4/XMM4	R12B/R12W/R12D/R12/ST4/MM4/XMM12		100	C4	CC	D4	DC	E4	EC	F4	FC	C4	CC	D4	DC	E4	EC	F4	FC
CH/BP/EBP/RBP/ST5/MM5/XMM5	R13B/R13W/R13D/R13/ST5/MM5/XMM13		101	C5	CD	D5	DD	E5	ED	F5	FD	C5	CD	D5	DD	E5	ED	F5	FD
DH/SI/ESI/RSI/ST6/MM6/XMM6	R14B/R14W/R14D/R14/ST6/MM6/XMM14		110	C6	CE	D6	DE	E6	EE	F6	FE	C6	CE	D6	DE	E6	EE	F6	FE
BH/DI/EDI/RDI/ST7/MM7/XMM7	R15B/R15W/R15D/R15/ST7/MM7/XMM15		111	C7	CF	D7	DF	E7	EF	F7	FF	C7	CF	D7	DF	E7	EF	F7	FF

94

New table: SIB

- http://ref.x86asm.net/coder64-abc.html#sib_byte_32_64

				REX.B=1															
r64				RAX	RCX	RDX	RBX	RSP	[1]	RSI	RDI	R8	R9	R10	R11	R12	[2]	R14	R15
r32				EAX	ECX	EDX	EBX	ESP	[1]	ESI	EDI	R8D	R9D	R10D	R11D	R12D	[2]	R14D	R15D
(In decimal) Base =				0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) Base =				000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Scaled Index	Scaled Index REX.X=1	SS	Index	Value of SIB Byte (in Hex)								Value of SIB Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	01	02	03	04	05	06	07	00	01	02	03	04	05	06	07
[RCX/ECX]	[R9/R9D]		001	08	09	0A	0B	0C	0D	0E	0F	08	09	0A	0B	0C	0D	0E	0F
[RDX/EDX]	[R10/R10D]		010	10	11	12	13	14	15	16	17	10	11	12	13	14	15	16	17
[RBX/EBX]	[R11/R11D]		011	18	19	1A	1B	1C	1D	1E	1F	18	19	1A	1B	1C	1D	1E	1F
none	[R12/R12D]		100	20	21	22	23	24	25	26	27	20	21	22	23	24	25	26	27
[RBP/EBP]	[R13/R13D]		101	28	29	2A	2B	2C	2D	2E	2F	28	29	2A	2B	2C	2D	2E	2F
[RSI/ESI]	[R14/R14D]		110	30	31	32	33	34	35	36	37	30	31	32	33	34	35	36	37
[RDI/EDI]	[R15/R15D]		111	38	39	3A	3B	3C	3D	3E	3F	38	39	3A	3B	3C	3D	3E	3F
[RAX/EAX*2]	[R8/R8D*2]	01	000	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
[RCX/ECX*2]	[R9/R9D*2]		001	48	49	4A	4B	4C	4D	4E	4F	48	49	4A	4B	4C	4D	4E	4F
[RDX/EDX*2]	[R10/R10D*2]		010	50	51	52	53	54	55	56	57	50	51	52	53	54	55	56	57
[RBX/EBX*2]	[R11/R11D*2]		011	58	59	5A	5B	5C	5D	5E	5F	58	59	5A	5B	5C	5D	5E	5F
none	[R12/R12D*2]		100	60	61	62	63	64	65	66	67	60	61	62	63	64	65	66	67
[RBP/EBP*2]	[R13/R13*2]		101	68	69	6A	6B	6C	6D	6E	6F	68	69	6A	6B	6C	6D	6E	6F
[RSI/ESI*2]	[R14/R14D*2]		110	70	71	72	73	74	75	76	77	70	71	72	73	74	75	76	77
[RDI/EDI*2]	[R15/R15D*2]		111	78	79	7A	7B	7C	7D	7E	7F	78	79	7A	7B	7C	7D	7E	7F
[RAX/EAX*4]	[R8/R8D*4]	10	000	80	81	82	83	84	85	86	87	80	81	82	83	84	85	86	87
[RCX/ECX*4]	[R9/R9D*4]		001	88	89	8A	8B	8C	8D	8E	8F	88	89	8A	8B	8C	8D	8E	8F
[RDX/EDX*4]	[R10/R10D*4]		010	90	91	92	93	94	95	96	97	90	91	92	93	94	95	96	97
[RBX/EBX*4]	[R11/R11D*4]		011	98	99	9A	9B	9C	9D	9E	9F	98	99	9A	9B	9C	9D	9E	9F
none	[R12/R12D*4]		100	A0	A1	A2	A3	A4	A5	A6	A7	A0	A1	A2	A3	A4	A5	A6	A7
[RBP/EBP*4]	[R13/R13D*4]		101	A8	A9	AA	AB	AC	AD	AE	AF	A8	A9	AA	AB	AC	AD	AE	AF
[RSI/ESI*4]	[R14/R14D*4]		110	B0	B1	B2	B3	B4	B5	B6	B7	B0	B1	B2	B3	B4	B5	B6	B7
[RDI/EDI*4]	[R15/R15D*4]		111	B8	B9	BA	BB	BC	BD	BE	BF	B8	B9	BA	BB	BC	BD	BE	BF
[RAX/EAX*8]	[R8/R8D*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7	C0	C1	C2	C3	C4	C5	C6	C7
[RCX/ECX*8]	[R9/R9D*8]		001	C8	C9	CA	CB	CC	CD	CE	CF	C8	C9	CA	CB	CC	CD	CE	CF
[RDX/EDX*8]	[R10/R10D*8]		010	D0	D1	D2	D3	D4	D5	D6	D7	D0	D1	D2	D3	D4	D5	D6	D7
[RBX/EBX*8]	[R11/R11D*8]		011	D8	D9	DA	DB	DC	DD	DE	DF	D8	D9	DA	DB	DC	DD	DE	DF
none	[R12/R12D*8]		100	E0	E1	E2	E3	E4	E5	E6	E7	E0	E1	E2	E3	E4	E5	E6	E7
[RBP/EBP*8]	[R13/R13D*8]		101	E8	E9	EA	EB	EC	ED	EE	EF	E8	E9	EA	EB	EC	ED	EE	EF
[RSI/ESI*8]	[R14/R14D*8]		110	F0	F1	F2	F3	F4	F5	F6	F7	F0	F1	F2	F3	F4	F5	F6	F7
[RDI/EDI*8]	[R15/R15D*8]		111	F8	F9	FA	FB	FC	FD	FE	FF	F8	F9	FA	FB	FC	FD	FE	FF

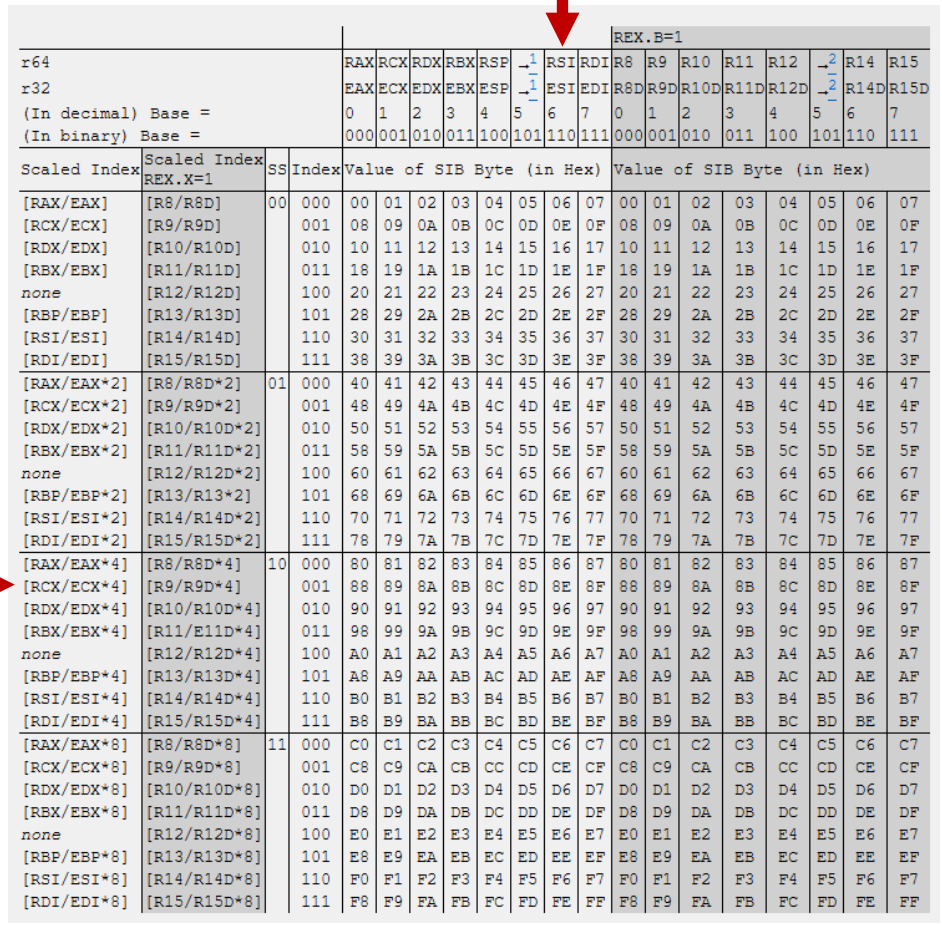
New table: SIB: [rsi+**rcx*4**+0x520],rdx

Pick the index RCX*4

													REX.B=1							
r64				RAX	RCX	RDX	RBX	RSP	→1	RSI	RDI	R8	R9	R10	R11	R12	→2	R14	R15
r32				EAX	ECX	EDX	EBX	ESP	→1	ESI	EDI	R8D	R9D	R10D	R11D	R12D	→2	R14D	R15D
(In decimal) Base =				0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) Base =				000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Scaled Index	Scaled Index REX.X=1	SS	Index	Value of SIB Byte (in Hex)								Value of SIB Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	01	02	03	04	05	06	07	00	01	02	03	04	05	06	07
[RCX/ECX]	[R9/R9D]		001	08	09	0A	0B	0C	0D	0E	0F	08	09	0A	0B	0C	0D	0E	0F
[RDX/EDX]	[R10/R10D]		010	10	11	12	13	14	15	16	17	10	11	12	13	14	15	16	17
[RBX/EBX]	[R11/R11D]		011	18	19	1A	1B	1C	1D	1E	1F	18	19	1A	1B	1C	1D	1E	1F
none	[R12/R12D]		100	20	21	22	23	24	25	26	27	20	21	22	23	24	25	26	27
[RBP/EBP]	[R13/R13D]		101	28	29	2A	2B	2C	2D	2E	2F	28	29	2A	2B	2C	2D	2E	2F
[RSI/ESI]	[R14/R14D]		110	30	31	32	33	34	35	36	37	30	31	32	33	34	35	36	37
[RDI/EDI]	[R15/R15D]		111	38	39	3A	3B	3C	3D	3E	3F	38	39	3A	3B	3C	3D	3E	3F
[RAX/EAX*2]	[R8/R8D*2]	01	000	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
[RCX/ECX*2]	[R9/R9D*2]		001	48	49	4A	4B	4C	4D	4E	4F	48	49	4A	4B	4C	4D	4E	4F
[RDX/EDX*2]	[R10/R10D*2]		010	50	51	52	53	54	55	56	57	50	51	52	53	54	55	56	57
[RBX/EBX*2]	[R11/R11D*2]		011	58	59	5A	5B	5C	5D	5E	5F	58	59	5A	5B	5C	5D	5E	5F
none	[R12/R12D*2]		100	60	61	62	63	64	65	66	67	60	61	62	63	64	65	66	67
[RBP/EBP*2]	[R13/R13*2]		101	68	69	6A	6B	6C	6D	6E	6F	68	69	6A	6B	6C	6D	6E	6F
[RSI/ESI*2]	[R14/R14D*2]		110	70	71	72	73	74	75	76	77	70	71	72	73	74	75	76	77
[RDI/EDI*2]	[R15/R15D*2]		111	78	79	7A	7B	7C	7D	7E	7F	78	79	7A	7B	7C	7D	7E	7F
[RAX/EAX*4]	[R8/R8D*4]	10	000	80	81	82	83	84	85	86	87	80	81	82	83	84	85	86	87
[RCX/ECX*4]	[R9/R9D*4]		001	88	89	8A	8B	8C	8D	8E	8F	88	89	8A	8B	8C	8D	8E	8F
[RDX/EDX*4]	[R10/R10D*4]		010	90	91	92	93	94	95	96	97	90	91	92	93	94	95	96	97
[RBX/EBX*4]	[R11/E11D*4]		011	98	99	9A	9B	9C	9D	9E	9F	98	99	9A	9B	9C	9D	9E	9F
none	[R12/R12D*4]		100	A0	A1	A2	A3	A4	A5	A6	A7	A0	A1	A2	A3	A4	A5	A6	A7
[RBP/EBP*4]	[R13/R13D*4]		101	A8	A9	AA	AB	AC	AD	AE	AF	A8	A9	AA	AB	AC	AD	AE	AF
[RSI/ESI*4]	[R14/R14D*4]		110	B0	B1	B2	B3	B4	B5	B6	B7	B0	B1	B2	B3	B4	B5	B6	B7
[RDI/EDI*4]	[R15/R15D*4]		111	B8	B9	BA	BB	BC	BD	BE	BF	B8	B9	BA	BB	BC	BD	BE	BF
[RAX/EAX*8]	[R8/R8D*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7	C0	C1	C2	C3	C4	C5	C6	C7
[RCX/ECX*8]	[R9/R9D*8]		001	C8	C9	CA	CB	CC	CD	CE	CF	C8	C9	CA	CB	CC	CD	CE	CF
[RDX/EDX*8]	[R10/R10D*8]		010	D0	D1	D2	D3	D4	D5	D6	D7	D0	D1	D2	D3	D4	D5	D6	D7
[RBX/EBX*8]	[R11/R11D*8]		011	D8	D9	DA	DB	DC	DD	DE	DF	D8	D9	DA	DB	DC	DD	DE	DF
none	[R12/R12D*8]		100	E0	E1	E2	E3	E4	E5	E6	E7	E0	E1	E2	E3	E4	E5	E6	E7
[RBP/EBP*8]	[R13/R13D*8]		101	E8	E9	EA	EB	EC	ED	EE	EF	E8	E9	EA	EB	EC	ED	EE	EF
[RSI/ESI*8]	[R14/R14D*8]		110	F0	F1	F2	F3	F4	F5	F6	F7	F0	F1	F2	F3	F4	F5	F6	F7
[RDI/EDI*8]	[R15/R15D*8]		111	F8	F9	FA	FB	FC	FD	FE	FF	F8	F9	FA	FB	FC	FD	FE	FF

[rsi+rcx*4+0x520],rdx

Pick the index RCX*4



												REX.B=1								
r64				RAX	RCX	RDX	RBX	RSP	→1	RSI	RDI	R8	R9	R10	R11	R12	→2	R14	R15
r32				EAX	ECX	EDX	EBX	ESP	→1	ESI	EDI	R8D	R9D	R10D	R11D	R12D	→2	R14D	R15D
(In decimal) Base =				0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) Base =				000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Scaled Index	Scaled Index REX.X=1	SS	Index	Value of SIB Byte (in Hex)								Value of SIB Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	01	02	03	04	05	06	07	00	01	02	03	04	05	06	07
[RCX/ECX]	[R9/R9D]		001	08	09	0A	0B	0C	0D	0E	0F	08	09	0A	0B	0C	0D	0E	0F
[RDX/EDX]	[R10/R10D]		010	10	11	12	13	14	15	16	17	10	11	12	13	14	15	16	17
[RBX/EBX]	[R11/R11D]		011	18	19	1A	1B	1C	1D	1E	1F	18	19	1A	1B	1C	1D	1E	1F
none	[R12/R12D]		100	20	21	22	23	24	25	26	27	20	21	22	23	24	25	26	27
[RBP/EBP]	[R13/R13D]		101	28	29	2A	2B	2C	2D	2E	2F	28	29	2A	2B	2C	2D	2E	2F
[RSI/ESI]	[R14/R14D]		110	30	31	32	33	34	35	36	37	30	31	32	33	34	35	36	37
[RDI/EDI]	[R15/R15D]		111	38	39	3A	3B	3C	3D	3E	3F	38	39	3A	3B	3C	3D	3E	3F
[RAX/EAX*2]	[R8/R8D*2]	01	000	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
[RCX/ECX*2]	[R9/R9D*2]		001	48	49	4A	4B	4C	4D	4E	4F	48	49	4A	4B	4C	4D	4E	4F
[RDX/EDX*2]	[R10/R10D*2]		010	50	51	52	53	54	55	56	57	50	51	52	53	54	55	56	57
[RBX/EBX*2]	[R11/R11D*2]		011	58	59	5A	5B	5C	5D	5E	5F	58	59	5A	5B	5C	5D	5E	5F
none	[R12/R12D*2]		100	60	61	62	63	64	65	66	67	60	61	62	63	64	65	66	67
[RBP/EBP*2]	[R13/R13*2]		101	68	69	6A	6B	6C	6D	6E	6F	68	69	6A	6B	6C	6D	6E	6F
[RSI/ESI*2]	[R14/R14D*2]		110	70	71	72	73	74	75	76	77	70	71	72	73	74	75	76	77
[RDI/EDI*2]	[R15/R15D*2]		111	78	79	7A	7B	7C	7D	7E	7F	78	79	7A	7B	7C	7D	7E	7F
[RAX/EAX*4]	[R8/R8D*4]	10	000	80	81	82	83	84	85	86	87	80	81	82	83	84	85	86	87
[RCX/ECX*4]	[R9/R9D*4]		001	88	89	8A	8B	8C	8D	8E	8F	88	89	8A	8B	8C	8D	8E	8F
[RDX/EDX*4]	[R10/R10D*4]		010	90	91	92	93	94	95	96	97	90	91	92	93	94	95	96	97
[RBX/EBX*4]	[R11/E11D*4]		011	98	99	9A	9B	9C	9D	9E	9F	98	99	9A	9B	9C	9D	9E	9F
none	[R12/R12D*4]		100	A0	A1	A2	A3	A4	A5	A6	A7	A0	A1	A2	A3	A4	A5	A6	A7
[RBP/EBP*4]	[R13/R13D*4]		101	A8	A9	AA	AB	AC	AD	AE	AF	A8	A9	AA	AB	AC	AD	AE	AF
[RSI/ESI*4]	[R14/R14D*4]		110	B0	B1	B2	B3	B4	B5	B6	B7	B0	B1	B2	B3	B4	B5	B6	B7
[RDI/EDI*4]	[R15/R15D*4]		111	B8	B9	BA	BB	BC	BD	BE	BF	B8	B9	BA	BB	BC	BD	BE	BF
[RAX/EAX*8]	[R8/R8D*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7	C0	C1	C2	C3	C4	C5	C6	C7
[RCX/ECX*8]	[R9/R9D*8]		001	C8	C9	CA	CB	CC	CD	CE	CF	C8	C9	CA	CB	CC	CD	CE	CF
[RDX/EDX*8]	[R10/R10D*8]		010	D0	D1	D2	D3	D4	D5	D6	D7	D0	D1	D2	D3	D4	D5	D6	D7
[RBX/EBX*8]	[R11/R11D*8]		011	D8	D9	DA	DB	DC	DD	DE	DF	D8	D9	DA	DB	DC	DD	DE	DF
none	[R12/R12D*8]		100	E0	E1	E2	E3	E4	E5	E6	E7	E0	E1	E2	E3	E4	E5	E6	E7
[RBP/EBP*8]	[R13/R13D*8]		101	E8	E9	EA	EB	EC	ED	EE	EF	E8	E9	EA	EB	EC	ED	EE	EF
[RSI/ESI*8]	[R14/R14D*8]		110	F0	F1	F2	F3	F4	F5	F6	F7	F0	F1	F2	F3	F4	F5	F6	F7
[RDI/EDI*8]	[R15/R15D*8]		111	F8	F9	FA	FB	FC	FD	FE	FF	F8	F9	FA	FB	FC	FD	FE	FF

Pick the displacement register, **RSI**

[rsi+rcx*4+0x520],rdx

Pick the index RCX*4

											REX.B=1							
r64				RAX	RCX	RDX	RBX	RSP	→[1]	RSI	RDI	R8	R9	R10	R11	R12	→[2]	R14	R15
r32				EAX	ECX	EDX	EBX	ESP	→[1]	ESI	EDI	R8D	R9D	R10D	R11D	R12D	→[2]	R14D	R15D
(In decimal) Base =				0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
(In binary) Base =				000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
Scaled Index	Scaled Index REX.X=1	SS	Index	Value of SIB Byte (in Hex)								Value of SIB Byte (in Hex)							
[RAX/EAX]	[R8/R8D]	00	000	00	01	02	03	04	05	06	07	00	01	02	03	04	05	06	07
[RCX/ECX]	[R9/R9D]		001	08	09	0A	0B	0C	0D	0E	0F	08	09	0A	0B	0C	0D	0E	0F
[RDX/EDX]	[R10/R10D]		010	10	11	12	13	14	15	16	17	10	11	12	13	14	15	16	17
[RBX/EBX]	[R11/R11D]		011	18	19	1A	1B	1C	1D	1E	1F	18	19	1A	1B	1C	1D	1E	1F
none	[R12/R12D]		100	20	21	22	23	24	25	26	27	20	21	22	23	24	25	26	27
[RBP/EBP]	[R13/R13D]		101	28	29	2A	2B	2C	2D	2E	2F	28	29	2A	2B	2C	2D	2E	2F
[RSI/ESI]	[R14/R14D]		110	30	31	32	33	34	35	36	37	30	31	32	33	34	35	36	37
[RDI/EDI]	[R15/R15D]		111	38	39	3A	3B	3C	3D	3E	3F	38	39	3A	3B	3C	3D	3E	3F
[RAX/EAX*2]	[R8/R8D*2]	01	000	40	41	42	43	44	45	46	47	40	41	42	43	44	45	46	47
[RCX/ECX*2]	[R9/R9D*2]		001	48	49	4A	4B	4C	4D	4E	4F	48	49	4A	4B	4C	4D	4E	4F
[RDX/EDX*2]	[R10/R10D*2]		010	50	51	52	53	54	55	56	57	50	51	52	53	54	55	56	57
[RBX/EBX*2]	[R11/R11D*2]		011	58	59	5A	5B	5C	5D	5E	5F	58	59	5A	5B	5C	5D	5E	5F
none	[R12/R12D*2]		100	60	61	62	63	64	65	66	67	60	61	62	63	64	65	66	67
[RBP/EBP*2]	[R13/R13*2]		101	68	69	6A	6B	6C	6D	6E	6F	68	69	6A	6B	6C	6D	6E	6F
[RSI/ESI*2]	[R14/R14D*2]		110	70	71	72	73	74	75	76	77	70	71	72	73	74	75	76	77
[RDI/EDI*2]	[R15/R15D*2]		111	78	79	7A	7B	7C	7D	7E	7F	78	79	7A	7B	7C	7D	7E	7F
[RAX/EAX*4]	[R8/R8D*4]	10	000	80	81	82	83	84	85	86	87	80	81	82	83	84	85	86	87
										8E	8F	88	89	8A	8B	8C	8D	8E	8F
[RDX/EDX*4]	[R10/R10D*4]		010	90	91	92	93	94	95	96	97	90	91	92	93	94	95	96	97
[RBX/EBX*4]	[R11/E11D*4]		011	98	99	9A	9B	9C	9D	9E	9F	98	99	9A	9B	9C	9D	9E	9F
none	[R12/R12D*4]		100	A0	A1	A2	A3	A4	A5	A6	A7	A0	A1	A2	A3	A4	A5	A6	A7
[RBP/EBP*4]	[R13/R13D*4]		101	A8	A9	AA	AB	AC	AD	AE	AF	A8	A9	AA	AB	AC	AD	AE	AF
[RSI/ESI*4]	[R14/R14D*4]		110	B0	B1	B2	B3	B4	B5	B6	B7	B0	B1	B2	B3	B4	B5	B6	B7
[RDI/EDI*4]	[R15/R15D*4]		111	B8	B9	BA	BB	BC	BD	BE	BF	B8	B9	BA	BB	BC	BD	BE	BF
[RAX/EAX*8]	[R8/R8D*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7	C0	C1	C2	C3	C4	C5	C6	C7
[RCX/ECX*8]	[R9/R9D*8]		001	C8	C9	CA	CB	CC	CD	CE	CF	C8	C9	CA	CB	CC	CD	CE	CF
[RDX/EDX*8]	[R10/R10D*8]		010	D0	D1	D2	D3	D4	D5	D6	D7	D0	D1	D2	D3	D4	D5	D6	D7
[RBX/EBX*8]	[R11/R11D*8]		011	D8	D9	DA	DB	DC	DD	DE	DF	D8	D9	DA	DB	DC	DD	DE	DF
none	[R12/R12D*8]		100	E0	E1	E2	E3	E4	E5	E6	E7	E0	E1	E2	E3	E4	E5	E6	E7
[RBP/EBP*8]	[R13/R13D*8]		101	E8	E9	EA	EB	EC	ED	EE	EF	E8	E9	EA	EB	EC	ED	EE	EF
[RSI/ESI*8]	[R14/R14D*8]		110	F0	F1	F2	F3	F4	F5	F6	F7	F0	F1	F2	F3	F4	F5	F6	F7
[RDI/EDI*8]	[R15/R15D*8]		111	F8	F9	FA	FB	FC	FD	FE	FF	F8	F9	FA	FB	FC	FD	FE	FF

Pick the displacement register, RSI

8E

mov [rsi+rcx*4+0x520],rdx

- Prefix: **48**
- Opcode: **89**
- Operand RDX with SIB: **94**
- Index RCX, Displacement RSI: **8E**

- Anything missing?

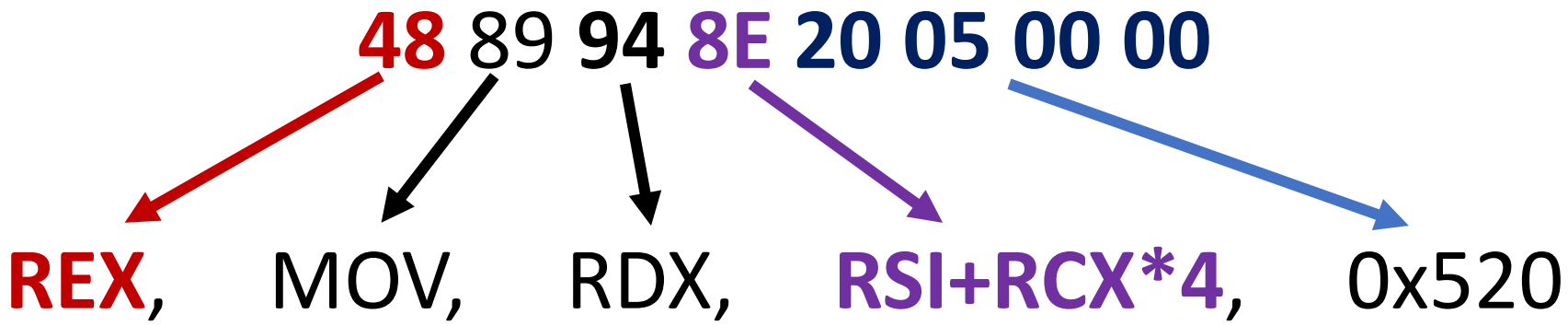
mov [rsi+rcx*4+0x520],rdx

- Prefix: 48
- Opcode: 89
- Operand RDX with SIB: 0x94
- Index RCX, Displacement RSI: 0x8E

- Last step: encode **0x0520**, we specifically are using "disp32" (4 bytes)

mov [rsi+rcx*4+0x520],rdx

- End Result:

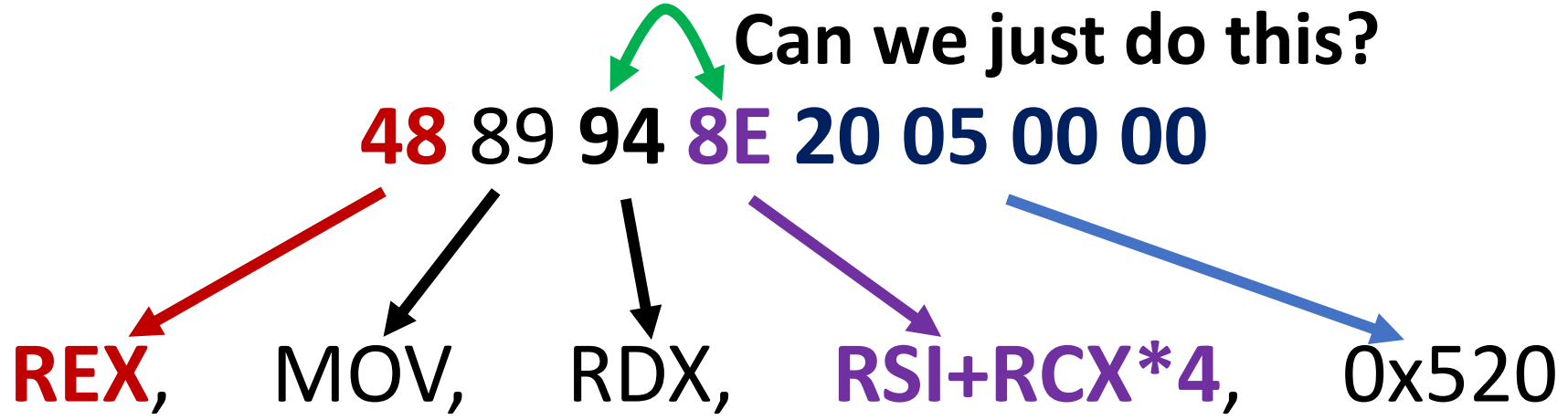


What about: mov rdx,[rsi+rcx*4+0x520]

- We want to read **from memory and store it in RDX**
- From what you have learned from previous ISAs, what would you think happens?

What about: mov rdx,[rsi+rcx*4+0x520]

- We want to read from memory and store it in RDX
- Intuitively, one could assume it just flips the order of the bytes.



mov rdx,[rsi+rcx*4+0x520]

- If we try: **48** 89 **8E** **94** **20 05 00 00**
- The output we get is:

mov [rsi+0x52094],rcx

Uh oh…

Register "RM" vs Register "R"

- "**RM**" – registers and memory operations can happen when your operand is "RM"
- "**R**" – only registers

- The opcode 0x89 is: `MOV rm, r`
- The opcode 0x8B is: `MOV r, rm`

- The order of operands is determined by the opcode!
- The type of operands is determined by the opcode!

- If we look at ADD: https://www.felixcloutier.com/x86/add
- There are r/m8: AL, CL, DL, BL, AH, CH, DH, BH
- There are r/m32: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- There are r/m64: [prefix needed] RAX, RCX, RDX,…

Second Look: [regB+regX*8+520],regW

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend operand (imm, or register “r”) to 64-bit

Second Look: [rax+rcx*8+520],rdx

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend operand (imm, or register “r”) to 64-bit

Second Look: [rax+rcx*8+520],r10

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend operand (imm, or register “r”) to 64-bit

Second Look: [r11+rcx*8+520],r10

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend new (imm, or register “r”) to 64-bit

Second Look: [rax+r8*8+520],edx

REX	Description
B	Use new registers for displacement register
X	Use new registers for index register
R	Use new registers for “r” (regW above)
W	Extend operand (imm, or register “r”) to 64-bit

Overview

- Registers are encoded with one register being "**rm**" that allows displacement and indexing a memory read
- Another register can be "**r**" which is always a plain register that resolves to some value

- Some trickier operations coming up: "**push X**" only takes one operand, so which is it? RM? R?

Shared Opcodes

Just like in MIPS

- Recall: https://www.cs.unc.edu/~montek/teaching/Comp541-Fall23/Lab8/MIPS_Green_Sheet.pdf
- In MIPS: ADD, JMP, NOR, OR, SLT, SHR, SHL…
- All used opcode 0x00

- x86 does something similar.
- Open: http://ref.x86asm.net/coder64.html#x81

Immediate Operands

- ADD, OR, ADC, SBB, AND, SUB, XOR, CMP all use 0x81:
  - instruction r/m64, **imm32**

- Similarly, 0x83 is: r/m64,**imm8**

Immediate Operands

- If you see: 0x81 **/0**, 0x81 **/1**, 0x81 **/2**, etc…
- It means your "r/m" field is your desination
- It means use register index "**r**" of 0 for /0, 1 for /1, …

Immediate Operands

- If you see: 0x81 /0, 0x81 /1, 0x81 /2, etc…
- It means your "r/m" field is your desination
- It means use register index "r" of 0 for /0, 1 for /1, …

Example: SUB is 0x81 /5

This means that if my register "r" is index 5 (**RBP**),
that means the operation is "subtract immediate"

sub r10,0x520

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)
- RegisterRM: R10 (index 2, REX.B=1)

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)
- RegisterRM: R10 (index 2, REX.B=1)

- Encode: 0x49 (REX W+B)

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)
- RegisterRM: R10 (index 2, REX.B=1)

- Encode: 0x49 (REX W+B)
- Opcode: 0x81

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)
- RegisterRM: R10 (index 2, REX.B=1)

- Encode: 0x49 (REX W+B)
- Opcode: 0x81
- Registers: 0xEA (rm= R10, r=RBP)

sub r10,0x520

- Opcode: 0x81 (4 byte immediate)
- RegisterR: RBP (5)
- RegisterRM: R10 (index 2, REX.B=1)

- Encode: 0x49 (REX W+B)
- Opcode: 0x81
- Registers: 0xEA (rm= R10, r=RBP)
- Immediate: 0x0520 (20 05 00 00)

Result of: sub r10,0x0520

49 81 EA 20 05 00 00

What's Left:

- We did some simple instruction encoding

- Coming up:
  - Encoding more difficult instructions (and patching)
  - System calls
  - Organizing the visitor model, stack framing, and "where is my variable in memory?"
  - Setting up the ELF header

Final notes

- We will be doing position-independent code (like a shared object)
- You do not have to do this

- So long as your compiler outputs a binary that I can run like "./a.out" then that is what matters.
- (And of course, output must be correct)

Final notes (2)

- We will be doing position-independent code (like a shared object)
- You do not have to do this

- So long as your compiler outputs a binary that I can run like "./a.out" then that is what matters.
- (And of course, output must be correct)

PA3 Due Tomorrow!

- Have a great weekend!

End