SMT-Based Binary Analysis

Möbius Strip Reverse Engineering

April 19, 2018

1 Python

Read the first chapter in the implementation manual, on Python. There are several exercises to reinforce your understanding of the concepts discussed therein.

Programming Exercise 1. Read the implementation manual from section 1.1.1 to section 1.2.4. Invoke Python on `./Python/Transform.py`, for example, `\python27\python.exe -m Python.Transform`. The test-cases at the bottom of the file will run and print error messages. You are finished when you have implemented the missing functionality and all tests pass.

Programming Exercise 2. Read the implementation manual from section 1.3 to section 1.4.3. Invoke Python on `./Python/Iterator.py`, for example, `\python27\python.exe -m Python.Iterator`. You are finished when you have implemented the missing functionality and all tests pass.

Reading Exercise 3. Read the implementation manual from section 1.5 until the end of the chapter. This is especially important for people coming from languages other than Python. If you are already comfortable programming in Python, peruse the table of contents for the rest of this chapter and read anything that calls your attention.

2 X86

Reading Exercise 4. Read the implementation manual from section 2 to section 2.4.

X86 MOD R/M, two binary coding schemes for memory expressions, are instrumental to encoding and decoding X86 machine code. You will complete exercises to ensure that you understand how MOD R/M expressions are encoded and decoded, as well as implement encoding and decoding in Python.

2.1 MOD R/M-16

Written Exercise 5. For the purported 16-bit memory expressions in table 1:

- If they are illegal, explain why.
- If they are legal:
  - Encode them into a sequence of bytes using the MOD R/M-16 encoding scheme. Use 000 for the MOD R/M.GGG field.
    - Some expressions can possibly be encoded in two different ways. Always use the shorter encoding.
  - Indicate whether the expression is using the default segment, i.e., whether a segment prefix is required.

Written Exercise 6. Given the bytes in table 2, decode them as MOD R/M-16 expressions. Though all byte sequences are three bytes long, the length of the actual MOD R/M-16 expression may be less than three bytes. Give the parts of the MOD R/M-16, the lengths, and the corresponding memory expressions.

Programming Exercise 7. Read sections 2.5 and section 2.5.1 in the implementation manual. Your exercises are to implement MOD R/M-16 encoding and decoding. The `ModRM16::Decode` method simply consumes the MOD R/M byte from the `stream` object, splits it into MOD R/M's three constituent fields, determines whether a displacement is specified, and consumes the displacement if so. It then sets the class properties within its object to these values. As this code is fairly simple, it is provided for you.

The `Interpret` method is responsible for determining the memory expression specified by the MOD R/M. The return value is a tuple of the components of the memory expression. You are responsible for completing this method and causing the tests to pass.

The `EncodeFromParts` method is the inverse of `Interpret`: given the components of a memory expression, set the fields of the MOD R/M object to the proper values. You are responsible for completing this method and causing the tests to pass.

The tests for encoding work as follows:

1. Generate random memory expression components.
2. Call the method `EncodeFromParts` to set the fields of the `ModRM16` object to the integer values corresponding to those components
3. Call the method `Interpret` to retrieve the memory expression components from the `ModRM16` object
4. Ensures that the components are equal (or at least compatible – i.e., it is acceptable if one expression had a displacement of `0` and the other one had a displacement of `None` or vice versa).

After the encoding test passes for a specific set of randomly-generated memory expression components, we then test the decoder component.

1. Encode the `ModRM16` object from the encoder test using the `Encode` method, and create a `X86ByteStream` object `s` based on those bytes.
2. Create a new `ModRM16` object `m2`.
3. Call `m2.Decode(s)` to decode the bytes.
4. Call `m2.Interpret` to retrieve the memory expression components from `m2`.
5. Ensure they match (or are compatible with, as above) the original memory expression components that we generated randomly.

Run the unit tests for `Tests.X86.TestModRM16`. You are finished when the `test_ModRM16` test passes.

2.2 MOD R/M-32

Written Exercise 8. For the purported 32-bit memory expressions in table 3:

- If they are illegal, explain why.
- If they are legal:
  - Encode them into a sequence of bytes using the MOD R/M-32 encoding scheme. Use 000 for the MOD R/M.GGG field.
    * Some expressions can possibly be encoded in two different ways. Always use the shorter encoding.
  - Indicate whether the expression is using the default segment, i.e., whether a segment prefix is required.

Written Exercise 9. Given the bytes in table 4, decode them as MOD R/M-32 expressions. Though all byte sequences are six bytes long, the length of the actual MOD R/M-32 expression may be less than six bytes. Give the parts of the MOD R/M-32, the lengths, and the corresponding memory expressions.

Programming Exercise 10. Read section 2.5.2 in the implementation manual. The description of the MOD R/M-32 tests is identical to those for MOD R/M-16. Run the unit tests for `Tests.X86.TestModRM32`. You are finished when the `test_ModRM32` test passes.

2.3 X86 Type-Checking

2.3.1 Type-Checking Operands

Written Exercise 11. For the X86 instructions listed in table 6, determine which of the abstract operand types in table 5 match them, and which prefixes are required for them to match.

Programming Exercise 12. Read the implementation manual from section 2.6 to section 2.9.1. Type-checking is tested with a suite of both positive and negative examples. For each category of type-checking rules, there is an abstract operand type that will cause that rule to be invoked, as well as a list of both positive examples (i.e., things that should match) and negative examples (i.e., things that should not match). For those that should match, the prefix requirements must match those specified in the test cases. Run the unit tests for `Tests.X86.TestX86TypeChecker`. You are finished when the eight test cases pass.

There is an additional test suite in `Tests.X86.TestX86TypeCheckerRandomly`. You should invoke it to ensure that it runs with no errors. No additional work should be required for this step.

2.4 Encoding X86 Instructions

Written Exercise 13. For the encodings and X86 instructions listed in tables 7 through 17, perform instruction encoding by hand using the steps laid out in the slides:

1. Copy the stem into the encoder context.
2. Type-check the X86 operands against the AOTs listed, as in exercise 11, to learn which prefixes are required.
3. Attend to the encoding method:
   - For Size Prefix encodings, add an OPSIZE prefix to the encoder context.
   - For ModRM Group encodings, set MOD R/M.GGG in the encoder context to the group number indicated.
4. Retrieve the AOTDL entry for each abstract operand type, as listed in table 5. Encode each of the operands, using the logic for the respective AOTDL entry as laid out in the slides and implementation manual.
5. Concatenate the instruction parts together to obtain the machine code.

Table 1: MOD R/M-16 Encoding Exercises

Expression	Legal?	MOD R/M-16 Encoding	Prefix?
`es:[bx]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`ds:[bx+si+1234h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`ss:[sp]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`fs:[si+80h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`es:[bx+di+0FF80h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`es:[5678h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`ds:[bp+si+0FF7Fh]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`gs:[cx+si+9h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`ss:[bp+di+5h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	
`cs:[bp]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \underbrace{\quad}_{\text{Displacement}}$	

Table 2: MOD R/M-16 Decoding Exercises

Bytes	MOD R/M-16	Length	Expression
`03 20 62`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`02 A5 E6`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`80 D5 43`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`45 F4 38`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`06 A4 40`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`07 D4 3B`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`81 EA A3`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`44 6A 3D`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`00 F9 A5`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		
`05 FC 6D`	$\underbrace{\quad}_{\text{MOD R/M}}\ \underbrace{\quad}_{\text{Displacement}}$ $\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}$		

Table 3: MOD R/M-32 Encoding Exercises

Expression	Legal?	MOD R/M-32 Encoding	Prefix?
`fs:[ecx]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`cs:[edx+5AB04044h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`es:[49E98581h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`ss:[esp]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`ds:[ebp]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`ds:[ebx*4]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`ss:[esp*4]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`cs:[edx+edi*4]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`es:[ebx+edx*8+0FFFFFFA2h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`ss:[ebp+eax*8+3Eh]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	
`fs:[ebp+edi*4+64783E89h]`		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\ \overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\ \underbrace{\qquad\qquad}_{\text{Displacement}}$	

Table 4: MOD R/M-32 Decoding Exercises

Bytes	MOD R/M-32	Length	Expression
40 0D E4 17 3D 2C	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
03 90 1E 7C 1A 48	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
45 17 F7 CE 42 5B	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
44 34 36 2C B3 06	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
04 13 08 17 36 C7	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
46 05 18 C7 04 9B	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
02 A4 9F FD B1 47	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
84 B9 E5 5C 4E 0F	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
82 27 80 48 78 C7	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		
47 1B E0 79 ED F6	MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		

Table 5: AOTDL Example Translations

AOT	AOTDL
`OAL`	`Exact(Gb(Al))`
`OAX`	`Exact(Gw(Ax))`
`OCS`	`Exact(SegReg(CS))`
`OSt0`	`Exact(FPUReg(ST0))`
`OIb`	`ImmEnc(Ib)`
`OIw`	`ImmEnc(Iw)`
`OOb`	`ImmEnc(MemExpr(DS,Mb))`
`OGb`	`GPart(Gb)`
`OGw`	`GPart(Gw)`
`OGd`	`GPart(Gd)`
`OSw`	`GPart(SegReg)`
`OCd`	`GPart(ControlReg)`
`ODd`	`GPart(DebugReg)`
`OPd`	`GPart(MMXReg)`
`OVq`	`GPart(XMMReg)`
`OEb`	`RegOrMem(Gb,Mb)`
`OEw`	`RegOrMem(Gw,Mw)`
`OEd`	`RegOrMem(Gd,Md)`
`OQq`	`RegOrMem(MMXReg,Mq)`
`OWdq`	`RegOrMem(XMMReg,Mdq)`
`ORw`	`RegOrMem(Gw,None)`
`OMb`	`RegOrMem(None,Mb)`
`ORdMb`	`RegOrMem(Gd,Mb)`
`OStN`	`RegOrMem(FPUReg(ST0),None)`
`OeAX`	`SizePrefix(Exact(Gw(Ax)),Exact(Gd(Eax)))`
`OGv`	`SizePrefix(GPart(Gw),GPart(Gd))`
`OIv`	`SizePrefix(ImmEnc(Iw),ImmEnc(Id))`
`OIbv`	`SizePrefix(SignedImm(Iw),SignedImm(Id))`
`OEv`	`SizePrefix(RegOrMem(Gw,Mw),RegOrMem(Gd,Md))`
`OM`	`AddrPrefix(RegOrMem(None,Mw),RegOrMem(None,Md))`
`OYb`	`AddrPrefix(Exact(Mem16(ES,Mb,Di,None,None)),` `Exact(Mem32(ES,Mb,Edi,None,0,None))`
`OXw`	`AddrPrefix(ExactSeg(Mem16(DS,Mw,Si,None,None)),` `ExactSeg(Mem32(DS,Mw,Esi,None,0,None))`

Table 6: Type-Checking Exercises

X86 Operand	Python Representation
`al`	`Gb(Al)`
`ax`	`Gw(Ax)`
`eax`	`Gd(Eax)`
`1`	`Ib(1)`
`2`	`Iw(2)`
`3`	`Id(3)`
`cs`	`SegReg(CS)`
`st0`	`FPUReg(ST0)`
`cr4`	`ControlReg(CR4)`
`dr7`	`DebugReg(DR7)`
`mm3`	`MMXReg(MM3)`
`xmm2`	`XMMReg(XMM2)`
`byte ptr [eax]`	`Mem32(DS,Mb,Eax,None,0,None)`
`word ptr [bx]`	`Mem16(DS,Mw,Bx,None,None)`
`byte ptr es:[di]`	`Mem16(ES,Mb,Di,None,None)`
`word ptr gs:[esi]`	`Mem32(GS,Mw,Esi,None,0,None)`
`word ptr cs:[si]`	`Mem16(CS,Mw,Si,None,None)`
`byte ptr fs:[0]`	`Mem32(FS,Mb,None,None,0,0)`

Table 7: X86 Encoding Exercise #1

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`9C`			`pushfd`
Python Object	`Instruction([],Pushfd)`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\underbrace{\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}}_{}$ $\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{MOD R/M} \quad \text{SIB}}$ $\overbrace{\qquad\qquad}_{\text{Displacement}}$		

Table 8: X86 Encoding Exercise #2

Encoding Method	Stem	Operand Types		X86 Instruction
Size Prefix	`9C`			`pushfw`
Python Object	`Instruction([],Pushfw)`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\underbrace{\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}}_{}$ $\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{MOD R/M} \quad \text{SIB}}$ $\overbrace{\qquad\qquad}_{\text{Displacement}}$		

Table 9: X86 Encoding Exercise #3

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`EC`	`OAL ODX`		`in al, dx`
Python Object	`Instruction([],In,Gb(Al),Gw(Dx))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\underbrace{\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}}_{}$ $\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{MOD R/M} \quad \text{SIB}}$ $\overbrace{\qquad\qquad}_{\text{Displacement}}$		

Table 10: X86 Encoding Exercise #4

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`AC`	`OXb`		`lodsb fs:[edi]`
Python Object	`Instruction([],Lodsb,Mem32(FS,Mb,Edi,None,0,None))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\underbrace{\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}}_{}$ $\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{MOD R/M} \quad \text{SIB}}$ $\overbrace{\qquad\qquad}_{\text{Displacement}}$		

Table 11: X86 Encoding Exercise #5

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`04`	`OAL OIb`		`add al, 1`
Python Object	`Instruction([],Add,Gb(Al),Ib(1))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\text{MOD R/M}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{SIB}}\overbrace{\text{Displacement}}$		

Table 12: X86 Encoding Exercise #6

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`C2`	`OIw`		`retn 4`
Python Object	`Instruction([],Ret,Iw(4))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\text{MOD R/M}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{SIB}}\overbrace{\text{Displacement}}$		

Table 13: X86 Encoding Exercise #7

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`A2`	`OOb OAL`		`mov [12345h], al`
Python Object	`Instruction([],Mov,Mem32(DS,Mb,None,None,0,0x12345),Gb(Al))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\text{MOD R/M}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{SIB}}\overbrace{\text{Displacement}}$		

Table 14: X86 Encoding Exercise #8

Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	`33`	`OGv OEv`		`xor eax, ebx`
Python Object	`Instruction([],Xor,Gd(Eax),Gd(Ebx))`			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\text{MOD R/M}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\text{SIB}}\overbrace{\text{Displacement}}$		

Table 15: X86 Encoding Exercise #9

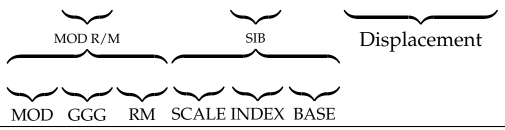
Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	33	OGv OEv		xor ax, bx
Python Object	Instruction([],Xor,Gw(Ax),Gw(Bx))			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		 MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		

Table 16: X86 Encoding Exercise #10

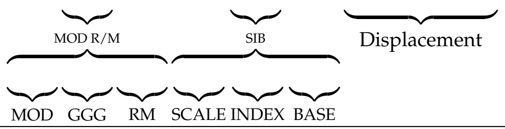
Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	68	OIv		push 12345h
Python Object	Instruction([],Push,Id(0x12345))			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		 MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		

Table 17: X86 Encoding Exercise #11

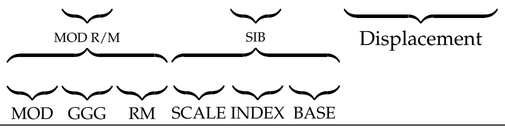
Encoding Method	Stem	Operand Types		X86 Instruction
Ordinary	6A	OIbv		push 0FF80h
Python Object	Instruction([],Push,Iw(0xFF80))			
Required Prefixes	Stem	MOD R/M (Optional)	Immediates	
		 MOD R/M SIB Displacement MOD GGG RM SCALE INDEX BASE		

Table 18: X86 Decoder Entries for Exercises

90	`Direct(Nop,[])`
9C	`PredOpSize(Direct(Pushfw,[]),Direct(Pushfd,[]))`
8F	`Group([Direct(Pop,[OEv]),Invalid*7])`
33	`Direct(Xor,[OGv,OEv])`
34	`Direct(Xor,[OAL,OIb])`
A2	`Direct(Mov,[OOb,OAL])`
AA	`Direct(Scasb,[OYb])`
0F 71	`PredMod(Invalid,Direct(Psrlw,[OQq,OIb]))`

Programming Exercise 14. Read sections 2.10 and section 2.10.1 in the implementation manual.

Encoding is tested by specifying an instruction with a known machine-code encoding, encoding it, and checking that the encoded byte sequences are equal. For each encoding method and category of encoding rules, an instruction has been chosen that will execute the relevant code paths. Run the unit tests for `Tests.X86.TestX86Encoder`. You are finished when the thirteen test cases pass.

There is an additional test suite in `Tests.X86.TestX86EncoderRandomly`. You should invoke it to ensure that it runs with no errors. No additional work should be required for this step.

2.5 Decoding X86 Instructions

Written Exercise 15. Decoding X86 instructions is accomplished in the opposite manner from encoding them. Specifically, you should look at the bytes and perform the following steps:

1. Consume all prefix bytes, and note their presence.
2. Consume the instruction stem, including any leading escape bytes (i.e., `0F`, `0F 38`, or `0F 3A`).
3. Look up the decoder entry for the instruction stem in table 18.
4. Process the decoder entry, as described in the slides. This may require inspecting prefixes, MOD R/M.GGG, MOD R/M.RM, MOD R/M.MOD, or some combination of those elements. At the end of this process, you should have a `Direct` decoder entry with a mnemonic and list of abstract operands.
5. Decode the abstract operands and turn them into actual X86 operands. Write the disassembled instruction the space indicated.

Programming Exercise 16. Read sections 2.11 and section 2.12 in the implementation manual.

Table 19: x86 Decoding Exercise #1

Bytes: `90 AE 82 14 BD E9 A8 56`

Required Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\underbrace{\qquad\qquad}_{\text{Displacement}}$		
Instruction:				

Table 20: x86 Decoding Exercise #2

Bytes: `66 9C 4C 99 42 0C C3 5F`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\underbrace{\qquad\qquad}_{\text{Displacement}}$		
Instruction:				

Table 21: x86 Decoding Exercise #3

Bytes: `66 8F C0 5F B6 07 B4 A4`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\underbrace{\qquad\qquad}_{\text{Displacement}}$		
Instruction:				

Table 22: x86 Decoding Exercise #4

Bytes: `8F 00 EC D3 60 E9 8B 04`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{\quad}_{\text{MOD}}\underbrace{\quad}_{\text{GGG}}\underbrace{\quad}_{\text{RM}}}^{\underbrace{\quad}_{\text{MOD R/M}}}\overbrace{\underbrace{\quad}_{\text{SCALE}}\underbrace{\quad}_{\text{INDEX}}\underbrace{\quad}_{\text{BASE}}}^{\underbrace{\quad}_{\text{SIB}}}\underbrace{\qquad\qquad}_{\text{Displacement}}$		
Instruction:				

Table 23: x86 Decoding Exercise #5

Bytes: `33 C0 12 AC A7 1C 3A D8`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		MOD R/M: MOD GGG RM; SIB: SCALE INDEX BASE; Displacement		
Instruction:				

Table 24: x86 Decoding Exercise #6

Bytes: `66 33 C0 B9 67 64 5E D1`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		MOD R/M: MOD GGG RM; SIB: SCALE INDEX BASE; Displacement		
Instruction:				

Table 25: x86 Decoding Exercise #7

Bytes: `34 FF 15 09 90 DD 9D D4`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		MOD R/M: MOD GGG RM; SIB: SCALE INDEX BASE; Displacement		
Instruction:				

Table 26: x86 Decoding Exercise #8

Bytes: `A2 E5 FB DB FF 93 2D CC`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		MOD R/M: MOD GGG RM; SIB: SCALE INDEX BASE; Displacement		
Instruction:				

Table 27: x86 Decoding Exercise #9

Bytes: `67 2E AA 95 4D 09 D4 D7`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{}_{\text{MOD}}\underbrace{}_{\text{GGG}}\underbrace{}_{\text{RM}}}^{\overbrace{}^{}\text{MOD R/M}}\ \overbrace{\underbrace{}_{\text{SCALE}}\underbrace{}_{\text{INDEX}}\underbrace{}_{\text{BASE}}}^{\overbrace{}^{}\text{SIB}}\ \overbrace{\text{Displacement}}$		
Instruction:				

Table 28: x86 Decoding Exercise #10

Bytes: `0F 71 00 F5 34 DE B1 41`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{}_{\text{MOD}}\underbrace{}_{\text{GGG}}\underbrace{}_{\text{RM}}}^{\overbrace{}^{}\text{MOD R/M}}\ \overbrace{\underbrace{}_{\text{SCALE}}\underbrace{}_{\text{INDEX}}\underbrace{}_{\text{BASE}}}^{\overbrace{}^{}\text{SIB}}\ \overbrace{\text{Displacement}}$		
Instruction:				

Table 29: x86 Decoding Exercise #11

Bytes: `0F 71 C0 F5 34 DE B1 41`

Prefixes	Stem	MOD R/M (Optional)	Immediates	Length
		$\overbrace{\underbrace{}_{\text{MOD}}\underbrace{}_{\text{GGG}}\underbrace{}_{\text{RM}}}^{\overbrace{}^{}\text{MOD R/M}}\ \overbrace{\underbrace{}_{\text{SCALE}}\underbrace{}_{\text{INDEX}}\underbrace{}_{\text{BASE}}}^{\overbrace{}^{}\text{SIB}}\ \overbrace{\text{Displacement}}$		
Instruction:				

The decoder is tested by specifying a list of machine code bytes whose decoding is known, decoding them, and then checking that the decoded instruction matches the expected result. For each type of DECDL decoder entry and category of decoding rules associated with AOTDL entries, an instruction has been chosen that will execute the relevant code paths. Run the unit tests for `Tests.X86.TestX86Decoder`. You are finished when the sixteen test cases pass.

After implementing the decoder, there is an extra test suite in `Tests.X86.TestX86EndToEnd` to test the entire X86 library. You should invoke it to ensure that it runs with no errors. No additional work should be required for this step.