D:/Coding/codeGen\1 & 2\main.py

```
# 1. Write a Python program that reads at least one of the YAML files in the RISC-V
#    Unified Database project (https://github.com/riscv-software-src/riscv-unified-db)
#    under spec/std/isa/inst.
import requests
import yaml
URL = "https://raw.githubusercontent.com/riscv-software-src/riscv-unified-db/main/spec/std/isa/inst/mock.yaml"
def main():
# Fetch the YAML file
response = requests.get(URL)
response.raise_for_status()
# Parse YAML
mock_data = yaml.safe_load(response.text)
print(yaml.dump(mock_data, sort_keys=False))
if __name__ == "__main__":
main()
# 2. Same Python program then emits the data in the YAML file as a C header file,
#    format of your choosing.
import requests
import yaml
import re
URL = "https://raw.githubusercontent.com/riscv-software-src/riscv-unified-db/main/spec/std/isa/inst/mock.yaml"
def sanitize_macro_name(name):
# Convert to uppercase and replace invalid characters with underscores
return re.sub(r'[^A-Z0-9_]', '_', name.upper())
def yaml_to_c_header(data):
lines = []
lines.append("/* Auto-generated from mock.yaml */")
lines.append("#ifndef MOCK_INSTRUCTION_H")
lines.append("#define MOCK_INSTRUCTION_H\n")
# Basic scalar fields
if "name" in data:
lines.append(f"#define INST_NAME \"{data['name']}\"")
if "long_name" in data:
lines.append(f"#define INST_LONG_NAME \"{data['long_name']}\"")
if "definedBy" in data:
lines.append(f"#define INST_DEFINED_BY \"{data['definedBy']}\"")
if "assembly" in data:
lines.append(f"#define INST_ASSEMBLY \"{data['assembly']}\"")
if "data_independent_timing" in data:
val = 1 if data["data_independent_timing"] else 0
lines.append(f"#define INST_DATA_INDEPENDENT_TIMING {val}")
```

```
# Encoding
if "encoding" in data:
enc = data["encoding"]
if "match" in enc:
lines.append(f"#define INST_ENCODING_MATCH \"{enc['match']}\"")
if "variables" in enc:
for var in enc["variables"]:
macro_name = sanitize_macro_name(f"INST_VAR_{var['name']}_LOC")
lines.append(f"#define {macro_name} \"{var['location']}\"")
# Access
if "access" in data:
for k, v in data["access"].items():
macro_name = sanitize_macro_name(f"INST_ACCESS_{k}")
lines.append(f"#define {macro_name} \"{v}\"")
lines.append("\n#endif /* MOCK_INSTRUCTION_H */")
return "\n".join(lines)
def main():
response = requests.get(URL)
response.raise_for_status()
mock_data = yaml.safe_load(response.text)
header_content = yaml_to_c_header(mock_data)
with open("mock_instruction.h", "w") as f:
f.write(header_content)
print("C header file generated: mock_instruction.h")
if __name__ == "__main__":
main()
```

D:/Coding/codeGen\1 & 2\mock_instruction.h

```
/* Auto-generated from mock.yaml */
#ifndef MOCK_INSTRUCTION_H
#define MOCK_INSTRUCTION_H
#define INST_NAME "mock"
#define INST_LONG_NAME "Mock Instruction (Just for testing UDB)"
#define INST_DEFINED_BY "Xmock"
#define INST_ASSEMBLY "xd, xs1, xs2"
#define INST_DATA_INDEPENDENT_TIMING 1
#define INST_ENCODING_MATCH "0000001----------000-----0001011"
#define INST_VAR_XS2_LOC "24-20"
#define INST_VAR_XS1_LOC "19-15"
#define INST_VAR_XD_LOC "11-7"
#define INST_ACCESS_S "always"
#define INST_ACCESS_U "always"
#define INST_ACCESS_VS "always"
#define INST_ACCESS_VU "always"
#endif /* MOCK_INSTRUCTION_H */
```

D:/Coding/codeGen\3\header.c

```
// 3. Write a C program that includes the C header file generated in step 2
#include
#include "mock_instruction.h"
int main(void) {
 printf("RISC-V Mock Instruction Info\n");
 printf("============================\n");
 printf("Name: %s\n", INST_NAME);
 printf("Long Name: %s\n", INST_LONG_NAME);
 printf("Defined By: %s\n", INST_DEFINED_BY);
 printf("Assembly Format: %s\n", INST_ASSEMBLY);
 printf("Data Independent Timing: %d\n", INST_DATA_INDEPENDENT_TIMING);
 printf("\nEncoding Match: %s\n", INST_ENCODING_MATCH);
 printf("Variable xs2 Location: %s\n", INST_VAR_XS2_LOC);
 printf("Variable xs1 Location: %s\n", INST_VAR_XS1_LOC);
 printf("Variable xd Location: %s\n", INST_VAR_XD_LOC);
 printf("\nAccess Permissions:\n");
 printf("  S: %s\n", INST_ACCESS_S);
 printf("  U: %s\n", INST_ACCESS_U);
 printf("  VS: %s\n", INST_ACCESS_VS);
 printf("  VU: %s\n", INST_ACCESS_VU);
 return 0;
}
```

D:/Coding/codeGen\3\mock_instruction.h

```
/* Auto-generated from mock.yaml */
#ifndef MOCK_INSTRUCTION_H
#define MOCK_INSTRUCTION_H
#define INST_NAME "mock"
#define INST_LONG_NAME "Mock Instruction (Just for testing UDB)"
#define INST_DEFINED_BY "Xmock"
#define INST_ASSEMBLY "xd, xs1, xs2"
#define INST_DATA_INDEPENDENT_TIMING 1
#define INST_ENCODING_MATCH "0000001----------000-----0001011"
#define INST_VAR_XS2_LOC "24-20"
#define INST_VAR_XS1_LOC "19-15"
#define INST_VAR_XD_LOC "11-7"
#define INST_ACCESS_S "always"
#define INST_ACCESS_U "always"
#define INST_ACCESS_VS "always"
#define INST_ACCESS_VU "always"
#endif /* MOCK_INSTRUCTION_H */
```

D:/Coding/codeGen\4\step4Header.c

```
// 4. Same C program should emit the contents of the C header file in YAML. The
//    YAML file does NOT need to match the original YAML file.
#include
#include "mock_instruction.h"
int main(void) {
 printf("# Instruction info in YAML format\n");
 printf("instruction:\n");
 printf("  name: \"%s\"\n", INST_NAME);
 printf("  long_name: \"%s\"\n", INST_LONG_NAME);
 printf("  defined_by: \"%s\"\n", INST_DEFINED_BY);
 printf("  assembly: \"%s\"\n", INST_ASSEMBLY);
 printf("  data_independent_timing: %s\n", INST_DATA_INDEPENDENT_TIMING ? "true" : "false");
 printf("  encoding:\n");
 printf("    match: \"%s\"\n", INST_ENCODING_MATCH);
 printf("    variables:\n");
 printf("      xs2: \"%s\"\n", INST_VAR_XS2_LOC);
 printf("      xs1: \"%s\"\n", INST_VAR_XS1_LOC);
 printf("      xd: \"%s\"\n", INST_VAR_XD_LOC);
 printf("  access:\n");
 printf("    s: \"%s\"\n", INST_ACCESS_S);
 printf("    u: \"%s\"\n", INST_ACCESS_U);
 printf("    vs: \"%s\"\n", INST_ACCESS_VS);
 printf("    vu: \"%s\"\n", INST_ACCESS_VU);
 return 0;
}
```

D:/Coding/codeGen\5\generated.yaml

```
$schema: inst_schema.json#
kind: instruction
name: mock
long_name: Mock Instruction (Just for testing UDB)
description: 'The mock instruction computes the value of PI to an infinite number
 of decimal places.
 Okay, actually it performs the equivalent of the `mul` instruction.
 [NOTE]
 Computing PI to an infinite number of decicial places is impossible, but hey, why
 not?
 '
definedBy: Xmock
assembly: xd, xs1, xs2
encoding:
 match: 0000001----------000-----0001011
 variables:
 - name: xs2
 location: 24-20
 - name: xs1
 location: 19-15
 - name: xd
 location: 11-7
access:
 s: always
 u: always
 vs: always
 vu: always
data_independent_timing: true
operation(): "#anchor(\"illegal-inst-exc-misa-disabled\") {\n if (implemented?(ExtensionName::M)\
 \ && (CSR[misa].M == 1'b0)) {\n raise (ExceptionCode::IllegalInstruction, mode(),\
 \ $encoding);\n }\n#}\n\nXReg src1 = X[xs1];\nXReg src2 = X[xs2];\n\n#anchor(\"\
 calculation\") {\n X[xd] = (src1 * src2)[MXLEN-1:0];\n#}\n"
sail(): "{\n if extension(\"M\") | haveZmmul() then {\n let rs1_val = X(rs1);\n\
 \  let rs2_val = X(rs2);\n let rs1_int : int = if signed1 then signed(rs1_val)\
 \ else unsigned(rs1_val);\n let rs2_int : int = if signed2 then signed(rs2_val)\
 \ else unsigned(rs2_val);\n let result_wide = to_bits(2 * sizeof(xlen), rs1_int\
 \ * rs2_int);\n let result = if high\n then result_wide[(2\
 \ * sizeof(xlen) - 1) .. sizeof(xlen)]\n else result_wide[(sizeof(xlen)\
 \ - 1) .. 0];\n X(rd) = result;\n RETIRE_SUCCESS\n } else {\n handle_illegal();\n\
 \  RETIRE_FAIL\n }\n}\n"
cert_normative_rules:
```

```
- id: inst.mock.encoding&basic;_op
 name: Encoding and basic operation
 description: Encoding and basic operation for `mock` instruction
 doc_links:
 - manual:inst:mul:encoding
 - udb:doc:inst:mock
- id: inst.mock.ill_exc_misa_M_disabled
 name: Illegal instruction exception when misa.M is 0
 description: 'An illegal instruction exception is raised when the instruction is
 executed
 and `misa.M` is 0.
 '
 doc_links:
 - manual:csr:misa:disabling-extension
cert_test_procedures:
- id: inst.mock.enc_and_basic
 description: Verify the encoding and basic operation of the `mock` instruction
 normative_rules:
 - inst.mock.encoding&basic;_op
 steps: '. Setup
 .. Load a variety of known values into rs1 & rs2 with a variety of rs1/rs2/rd
 values.
 . Execution
 .. Execute the `mock` instruction
 . Validation
 .. Check each result in rd
 . Teardown
 .. Clear the registers used for rd
 [NOTE]
 Don''t really need to clear the registers so this is a contrived example.
 I''ve got this note after the ordered list above.
 '
```

D:/Coding/codeGen\5\mock_instruction.h

```
/* Auto-generated from YAML */
#ifndef MOCK_INSTRUCTION_H
#define MOCK_INSTRUCTION_H
#define INST_NAME "mock"
#define INST_LONG_NAME "Mock Instruction (Just for testing UDB)"
#define INST_DEFINED_BY "Xmock"
#define INST_ASSEMBLY "xd, xs1, xs2"
#define INST_DATA_INDEPENDENT_TIMING 1
#define INST_ENCODING_MATCH "0000001----------000-----0001011"
#define INST_VAR_XS2_LOC "24-20"
#define INST_VAR_XS1_LOC "19-15"
#define INST_VAR_XD_LOC "11-7"
#define INST_ACCESS_S "always"
#define INST_ACCESS_U "always"
#define INST_ACCESS_VS "always"
#define INST_ACCESS_VU "always"
#endif /* MOCK_INSTRUCTION_H */
```

D:/Coding/codeGen\5\mock_out.yaml


```
$schema: inst_schema.json#
kind: instruction
name: mock
long_name: Mock Instruction (Just for testing UDB)
description: 'The mock instruction computes the value of PI to an infinite number
 of decimal places.
 Okay, actually it performs the equivalent of the `mul` instruction.
 [NOTE]
 Computing PI to an infinite number of decicial places is impossible, but hey, why
 not?
 '
definedBy: Xmock
assembly: xd, xs1, xs2
encoding:
 match: 0000001----------000-----0001011
 variables:
 - name: xs2
 location: 24-20
 - name: xs1
 location: 19-15
 - name: xd
 location: 11-7
access:
 s: always
 u: always
 vs: always
 vu: always
data_independent_timing: true
operation(): "#anchor(\"illegal-inst-exc-misa-disabled\") {\n if (implemented?(ExtensionName::M)\
 \ && (CSR[misa].M == 1'b0)) {\n raise (ExceptionCode::IllegalInstruction, mode(),\
 \ $encoding);\n }\n#}\n\nXReg src1 = X[xs1];\nXReg src2 = X[xs2];\n\n#anchor(\"\
 calculation\") {\n X[xd] = (src1 * src2)[MXLEN-1:0];\n#}\n"
sail(): "{\n if extension(\"M\") | haveZmmul() then {\n let rs1_val = X(rs1);\n\
 \  let rs2_val = X(rs2);\n let rs1_int : int = if signed1 then signed(rs1_val)\
 \ else unsigned(rs1_val);\n let rs2_int : int = if signed2 then signed(rs2_val)\
 \ else unsigned(rs2_val);\n let result_wide = to_bits(2 * sizeof(xlen), rs1_int\
 \ * rs2_int);\n let result = if high\n then result_wide[(2\
 \ * sizeof(xlen) - 1) .. sizeof(xlen)]\n else result_wide[(sizeof(xlen)\
 \ - 1) .. 0];\n X(rd) = result;\n RETIRE_SUCCESS\n } else {\n handle_illegal();\n\
 \  RETIRE_FAIL\n }\n}\n"
cert_normative_rules:
```

```
- id: inst.mock.encoding&basic;_op
 name: Encoding and basic operation
 description: Encoding and basic operation for `mock` instruction
 doc_links:
 - manual:inst:mul:encoding
 - udb:doc:inst:mock
- id: inst.mock.ill_exc_misa_M_disabled
 name: Illegal instruction exception when misa.M is 0
 description: 'An illegal instruction exception is raised when the instruction is
 executed
 and `misa.M` is 0.
 '
 doc_links:
 - manual:csr:misa:disabling-extension
cert_test_procedures:
- id: inst.mock.enc_and_basic
 description: Verify the encoding and basic operation of the `mock` instruction
 normative_rules:
 - inst.mock.encoding&basic;_op
 steps: '. Setup
 .. Load a variety of known values into rs1 & rs2 with a variety of rs1/rs2/rd
 values.
 . Execution
 .. Execute the `mock` instruction
 . Validation
 .. Check each result in rd
 . Teardown
 .. Clear the registers used for rd
 [NOTE]
 Don''t really need to clear the registers so this is a contrived example.
 I''ve got this note after the ordered list above.
 '
```

D:/Coding/codeGen\5\step5.c

```
#include
#include "mock_instruction.h"
int main(void) {
 printf("instruction:\n");
 printf("  name: \"%s\"\n", INST_NAME);
 printf("  long_name: \"%s\"\n", INST_LONG_NAME);
 printf("  defined_by: \"%s\"\n", INST_DEFINED_BY);
 printf("  assembly: \"%s\"\n", INST_ASSEMBLY);
 printf("  data_independent_timing: %s\n", INST_DATA_INDEPENDENT_TIMING ? "true" : "false");
 printf("  encoding:\n");
 printf("    match: \"%s\"\n", INST_ENCODING_MATCH);
 printf("    variables:\n");
 printf("      xs2: \"%s\"\n", INST_VAR_XS2_LOC);
 printf("      xs1: \"%s\"\n", INST_VAR_XS1_LOC);
 printf("      xd: \"%s\"\n", INST_VAR_XD_LOC);
 printf("  access:\n");
 printf("    s: \"%s\"\n", INST_ACCESS_S);
 printf("    u: \"%s\"\n", INST_ACCESS_U);
 printf("    vs: \"%s\"\n", INST_ACCESS_VS);
 printf("    vu: \"%s\"\n", INST_ACCESS_VU);
 return 0;
}
```

D:/Coding/codeGen\5\step5.py

```
import os
import re
import requests
import yaml
INPUT_YAML = "mock_out.yaml"
HEADER_FILE = "mock_instruction.h"
OUTPUT_YAML = "generated.yaml"
URL = "https://raw.githubusercontent.com/riscv-software-src/riscv-unified-db/main/spec/std/isa/inst/mock.yaml"
def ensure_yaml_exists():
if not os.path.exists(INPUT_YAML):
print(f"Downloading {INPUT_YAML} from GitHub...")
r = requests.get(URL)
r.raise_for_status()
with open(INPUT_YAML, "w", encoding="utf-8") as f:
f.write(r.text)
def sanitize_macro_name(name):
return re.sub(r'[^A-Z0-9_]', '_', name.upper())
def yaml_to_header_and_yaml(data):
lines_h = []
lines_h.append("/* Auto-generated from YAML */")
lines_h.append("#ifndef MOCK_INSTRUCTION_H")
lines_h.append("#define MOCK_INSTRUCTION_H\n")
lines_h.append(f"#define INST_NAME \"{data['name']}\"")
lines_h.append(f"#define INST_LONG_NAME \"{data['long_name']}\"")
lines_h.append(f"#define INST_DEFINED_BY \"{data['definedBy']}\"")
lines_h.append(f"#define INST_ASSEMBLY \"{data['assembly']}\"")
lines_h.append(f"#define INST_DATA_INDEPENDENT_TIMING {1 if data['data_independent_timing'] else 0}")
lines_h.append(f"#define INST_ENCODING_MATCH \"{data['encoding']['match']}\"")
for var in data['encoding']['variables']:
lines_h.append(f"#define INST_VAR_{sanitize_macro_name(var['name'])}_LOC \"{var['location']}\"")
for k, v in data['access'].items():
lines_h.append(f"#define INST_ACCESS_{k.upper()} \"{v}\"")
lines_h.append("\n#endif /* MOCK_INSTRUCTION_H */")
with open(HEADER_FILE, "w", encoding="utf-8") as f:
f.write("\n".join(lines_h))
with open(OUTPUT_YAML, "w", encoding="utf-8") as f:
yaml.safe_dump(data, f, sort_keys=False)
def main():
ensure_yaml_exists()
with open(INPUT_YAML, "r", encoding="utf-8") as f:
data = yaml.safe_load(f)
```

```
yaml_to_header_and_yaml(data)

if __name__ == "__main__":

main()
```