STUNIR OS-Level Code Generation

Overview

STUNIR now supports OS-level code generation for x86/x86_64 architectures. This includes:

- **Multiboot2 Header Generation**: Compliant bootloader headers for GRUB and other Multiboot2-compliant bootloaders
- **ISR (Interrupt Service Routine) Stub Generation**: Interrupt handling stubs for exceptions (0-31) and IRQs (32-255)
- **GDT (Global Descriptor Table) Generation**: Segment descriptors for memory segmentation
- **IDT (Interrupt Descriptor Table) Setup**: Interrupt descriptor table initialization

All generated code is **DO-178C Level A compliant**.

Quick Start

Python API

```
from targets.assembly.x86.multiboot2 import generate_multiboot2_header
from targets.assembly.x86.isr_stubs import generate_isr_stubs
from targets.assembly.x86.gdt_idt import generate_gdt, generate_idt_setup

# Generate Multiboot2 header
mb2_header = generate_multiboot2_header(arch='i386', syntax='intel')

# Generate ISR stubs
isr_stubs = generate_isr_stubs(mode='x86_64', syntax='intel', irq_count=16)

# Generate GDT
gdt_code = generate_gdt(mode='x86_64', syntax='intel')

# Generate IDT setup
idt_setup = generate_idt_setup(mode='x86_64', syntax='intel')
```

Using the X86 Emitter

```
from targets.assembly.x86.emitter import X86Emitter

ir_data = {...}  # Your STUNIR IR

emitter = X86Emitter(ir_data, 'output/', {
    '64bit': True,
    'multiboot2': True,
    'isr_stubs': True,
    'gdt': True,
    'idt': True
})

# Access individual generation methods
mb2 = emitter.emit_multiboot2_header()
isr = emitter.emit_isr_stubs()
gdt = emitter.emit_gdt()
idt = emitter.emit_idt_setup()
```

Multiboot2 Header Generation

Features

- Magic number: `0xE85250D6`
- Architecture support: i386, x86_64
- Optional tags:
- Information request (memory map)
- Framebuffer request
- Module alignment
- Automatic checksum calculation
- Boot entry point generation

Configuration Options

```
from targets.assembly.x86.multiboot2 import Multiboot2Config, Multiboot2HeaderGenerat-
or

config = Multiboot2Config(
    arch='i386',              # or 'x86_64'
    entry_point='_start',
    request_memory_map=True,
    request_framebuffer=False,
    fb_width=800,
    fb_height=600,
    fb_depth=32,
    syntax='intel'            # or 'att'
)

gen = Multiboot2HeaderGenerator(config)
header = gen.generate_header_asm()
boot_entry = gen.generate_boot_entry()
```

Example Output

```
; STUNIR Generated Multiboot2 Header
; DO-178C Level A Compliant

section .multiboot2
align 8

multiboot2_header_start:
    dd 0xE85250D6  ; magic
    dd 0           ; architecture (i386)
    dd multiboot2_header_end - multiboot2_header_start  ; length
    dd -(0xE85250D6 + 0 + (multiboot2_header_end - multiboot2_header_start))  ; check-
sum

; End Tag
align 8
end_tag:
    dw 0   ; type (end)
    dw 0   ; flags
    dd 8   ; size

multiboot2_header_end:
```

ISR Stub Generation

Features

- CPU exceptions (vectors 0-31)
- Hardware IRQs (configurable base, default 32-47)
- Automatic error code handling
- Common stub for register save/restore
- Support for both 32-bit and 64-bit modes

Exceptions with Error Codes

The following exceptions push an error code automatically:
- 8: Double Fault
- 10: Invalid TSS
- 11: Segment Not Present
- 12: Stack-Segment Fault
- 13: General Protection Fault
- 14: Page Fault
- 17: Alignment Check
- 21: Control Protection Exception
- 29: VMM Communication Exception
- 30: Security Exception

Configuration Options

```
from targets.assembly.x86.isr_stubs import ISRConfig, ISRStubGenerator

config = ISRConfig(
    mode='x86_64',                    # or 'x86_32'
    syntax='intel',                    # or 'att'
    exception_count=32,
    irq_base=32,
    irq_count=16,
    common_handler='interrupt_handler',
    save_all_regs=True
)

gen = ISRStubGenerator(config)
stubs = gen.generate_all_stubs()
idt_data = gen.generate_idt_table_data()
```

GDT Generation

Default Entries

Selector	Name	Description
0x00	null	Null descriptor (required)
0x08	kernel_code	Kernel code segment (Ring 0)
0x10	kernel_data	Kernel data segment (Ring 0)
0x18	user_code	User code segment (Ring 3)
0x20	user_data	User data segment (Ring 3)

Custom GDT Entries

```
from targets.assembly.x86.gdt_idt import GDTEntry, GDTConfig, GDTGenerator, GDTAccess,
GDTFlags

custom_entries = [
    GDTEntry(name="null"),
    GDTEntry(
        name="kernel_code",
        access=GDTAccess.PRESENT | GDTAccess.DESCRIPTOR | GDTAccess.EXECUTABLE |
GDTAccess.RW,
        flags=GDTFlags.LONG_MODE | GDTFlags.GRANULARITY
    ),
    # Add more entries...
]

config = GDTConfig(mode='x86_64', syntax='intel', entries=custom_entries)
gen = GDTGenerator(config)
gdt = gen.generate_gdt_asm()
```

IDT Setup Generation

Configuration Options

```
from targets.assembly.x86.gdt_idt import IDTConfig, IDTGenerator

config = IDTConfig(
    mode='x86_64',
    syntax='intel',
    vector_count=256,
    default_handler='default_interrupt_handler'
)

gen = IDTGenerator(config)
idt_setup = gen.generate_idt_setup_asm()
```

Ada SPARK Implementation

An Ada SPARK implementation is available for formal verification:

```
with X86_OS_Emitter;
use X86_OS_Emitter;

-- Generate Multiboot2 header
Emit_Multiboot2_Header (Content, Default_MB2_Config, Status);

-- Generate ISR common stub
Emit_ISR_Common_Stub (Content, Default_ISR_Config, Status);

-- Generate GDT
Emit_GDT (Content, Default_GDT_Config, Status);

-- Generate complete boot assembly
Emit_Complete_Boot_Assembly (Content, MB2_Cfg, ISR_Cfg, GDT_Cfg, Status);
```

Specification Templates

Templates for OS development are available in `spec/minimal_os/templates/`:

- `multiboot2_template.json` - Multiboot2 header specification
- `gdt_template.json` - GDT configuration
- `idt_template.json` - IDT configuration with all 32 exceptions and 16 IRQs

JSON schemas for validation are in `spec/minimal_os/schemas/`.

Building a Minimal OS

1. Generate Bootloader

```
python3 -c "
from targets.assembly.x86.multiboot2 import generate_multiboot2_header,
Multiboot2Config, Multiboot2HeaderGenerator
config = Multiboot2Config(arch='i386', syntax='intel')
gen = Multiboot2HeaderGenerator(config)
print(gen.generate_header_asm())
print(gen.generate_boot_entry())
" > boot.asm
```

2. Generate Interrupt Handling

```
python3 -c "
from targets.assembly.x86.isr_stubs import generate_isr_stubs
print(generate_isr_stubs(mode='x86_32', syntax='intel'))
" > isr.asm
```

3. Generate GDT

```
python3 -c "
from targets.assembly.x86.gdt_idt import generate_gdt
print(generate_gdt(mode='x86_32', syntax='intel'))
" > gdt.asm
```

4. Compile with NASM

```
nasm -f elf32 boot.asm -o boot.o
nasm -f elf32 isr.asm -o isr.o
nasm -f elf32 gdt.asm -o gdt.o
```

5. Link with your kernel

```
ld -m elf_i386 -T linker.ld -o kernel.elf boot.o isr.o gdt.o kernel.o
```

References

- Multiboot2 Specification (https://www.gnu.org/software/grub/manual/multiboot2/)
- Intel 64 and IA-32 Architectures Software Developer Manuals (https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html)
- OSDev Wiki (https://wiki.osdev.org/)