ECE 3574: Design Patterns and Idioms

Changwoo Min

Milestone 2

- **Develop a MIPS assembly simulator and its text-mode interface**
- **Three main components**
  - Lexer: the same as milestone 0
  - Parser: need to extend to produce information for VirtualMachine
    - initial memory for .data
    - program instructions for .text
    - labels for .data or .text
  - Virtual machine: our MIPS assembly simulator

Virtual Machine

1. MIPS registers (0-31, `pc`, `hi`, and `lo`)
2. Memory for .data (a sequence of bytes, 512 bytes)
3. Association between labels and memory locations or instructions
4. The program, as a sequence of insturctions

Example 1: after parsing

Assembly

```
        .data
var:    .word  305419896
           # 0x12345678
var2:   .word 3405692606
           # 0xcafebebe

        .text
main:
        lw $t0, var
        lw $t1, var2
end:
        j end
```

Initial memory

Data addr	Byte
0	0x78
1	0x56
2	0x34
3	0x12
4	0xbe
5	0xbe
6	0xfe
7	0xca

Instructions

Code addr	Instruction
0	lw $t0, var
1	lw $t1, var2
2	j end

Labels

Label	Type	Addr
var	.data	0
var2	.data	4
main	.text	0
end	.text	2

Example 1: initial status of Virtual Machine

Data memory

Data addr	Byte
0	0x78
1	0x56
2	0x34
3	0x12
4	0xbe
5	0xbe
6	0xfe
7	0xca

Instructions

Code addr	Instruction
0	lw $t0, var
1	lw $t1, var2
2	j end

Labels

Label	Type	Addr
var	.data	0
var2	.data	4
main	.text	0
end	.text	2

Registers

Register	Value
$pc	0
$t0	0
$t1	...

Assembly

```
       .data
var:   .word  305419896
          # 0x12345678
var2:  .word 3405692606
          # 0xcafebebe

       .text
main:
       lw $t0, var
       lw $t1, var2
end:
       j end
```

test01.asm: after parsing

Assembly

```
        .data
        .space 8
var1:   .word 1
var2:   .word -2

        .text
main:
      la $t0, var1

      lw $t1, 0
      lw $t2, $t0
      lw $t3, 4($t0)
      lw $t4, 4(var1)
      lw $t5, var2
```

Initial memory

Data addr	Byte
0	0x0
1	0x0
2	0x0
3	0x0
4	0x0
5	0x0
6	0x0
7	0x0
8	0x1
9	0x0
10	0x0
11	0x0
12	0xFE
13	0xFF
14	0xFF
15	0xFF

Instructions

Code addr	Instruction
0	la $t0, var1
1	lw $t1, 0
2	lw $t2, $t0
3	lw $t3, 4($t0)
4	lw $t4, 4(var1)
5	lw $t5, var2

Labels

Label	Type	Addr
var1	.data	8
var2	.data	12
main	.text	0

test01.asm: initial status of Virtual Machine

Data memory

Data addr	Byte
0	0x0
1	0x0
2	0x0
3	0x0
4	0x0
5	0x0
6	0x0
7	0x0
8	0x1
9	0x0
10	0x0
11	0x0
12	0xFE
13	0xFF
14	0xFF
15	0xFF

Instructions

Code addr	Instruction
0	la $t0, var1
1	lw $t1, 0
2	lw $t2, $t0
3	lw $t3, 4($t0)
4	lw $t4, 4(var1)
5	lw $t5, var2

Labels

Label	Type	Addr
var1	.data	8
var2	.data	12
main	.text	0

Registers

Register	Value
$pc	0
$t0	0
$t1	0
$t2	0
$t3	0
$t4	0
$t5	0

Assembly

```
        .data
        .space 8
var1:   .word 1
var2:   .word -2

        .text
main:
      la $t0, var1

      lw $t1, 0
      lw $t2, $t0
      lw $t3, 4($t0)
      lw $t4, 4(var1)
      lw $t5, var2
```

test02.asm: after parsing

Assembly

```
        .data
r1:     .space 4
r2:     .space 12
r3:     .space 4
var:    .word 7

        .text
main:
      la $t0, r2
      lw $t1, var

      sw $t1, 0
      sw $t1, $t0
      sw $t1, 4($t0)
      sw $t1, 8(r2)
      sw $t1, r
```

Initial memory

Data addr	Byte
0	0x0
...	...
4	0x0
...	...
16	0x0
...	...
20	0x7
21	0x0
22	0x0
23	0x0

Instructions

Code addr	Instruction
0	`la $t0, r2`
1	`lw $t1, var`
2	`sw $t1, 0`
3	`sw $t1, $t0`
4	`sw $t1, 4($t0)`
5	`sw $t1, 8(r2)`
6	`sw $t1, r`

Labels

Label	Type	Addr
r1	.data	0
r2	.data	4
r3	.data	16
var	.data	20
main	.text	0

test02.asm: initial status of Virtual Machine

Data memory

Data addr	Byte
0	0x0
...	...
4	0x0
...	...
16	0x0
...	...
20	0x7
21	0x0
22	0x0
23	0x0

Instructions

Code addr	Instruction
0	`la $t0, r2`
1	`lw $t1, var`
2	`sw $t1, 0`
3	`sw $t1, $t0`
4	`sw $t1, 4($t0)`
5	`sw $t1, 8(r2)`
6	`sw $t1, r`

Labels

Label	Type	Addr
r1	.data	0
r2	.data	4
r3	.data	16
var	.data	20
main	.text	0

Registers

Register	Value
$pc	0
$t0	0
$t1	0

Assembly

```
        .data
r1:     .space 4
r2:     .space 12
r3:     .space 4
var:    .word 7

        .text
main:
      la $t0, r2
      lw $t1, var

      sw $t1, 0
      sw $t1, $t0
      sw $t1, 4($t0)
      sw $t1, 8(r2)
      sw $t1, r
```

test05.asm: after parsing

Assembly

```
        .data
VALUE = -1
var:    .word 1
        .text
main:
        lw $t0, var
        add $t1, $t0, VALUE
        add $t2, $t1, $t0
```

Initial memory

Data addr	Byte
0	0x1
1	0x0
2	0x0
3	0x0

Instructions

Code addr	Instruction
0	`lw $t0, var`
1	`add $t1, $t0, VALUE`
2	`add $t2, $t1, $t0`

Labels

Label	Type	Addr
var	.data	0
main	.text	0

Constant

Name	Value
VALUE	-1

test05.asm: initial status of Virtual Machine

Data memory

Data addr	Byte
0	0x1
1	0x0
2	0x0
3	0x0

Instructions

Code addr	Instruction
0	`lw $t0, var`
1	`add $t1, $t0, VALUE`
2	`add $t2, $t1, $t0`

Labels

Label	Type	Addr
var	.data	0
main	.text	0

Registers

Register	Value
$pc	0
$t0	0
$t1	0
$t2	0

Constant

Name	Value
VALUE	-1

Assembly

```
        .data
VALUE = -1
var:    .word 1
        .text
main:
        lw $t0, var
        add $t1, $t0, VALUE
        add $t2, $t1, $t0
```

test18.asm: after parsing

Assembly

```
        .data
        .text
main:
        nop
        j next
        nop
next:
        nop
        j main
```

Initial memory

Data addr	Byte

Instructions

Code addr	Instruction
0	`nop`
1	`j next`
2	`nop`
3	`nop`
4	`j main`

Labels

Label	Type	Addr
main	.text	0
next	.text	3

test18.asm: initial status of Virtual Machine

Data memory

Data addr	Byte

Instructions

Code addr	Instruction
0	`nop`
1	`j next`
2	`nop`
3	`nop`
4	`j main`

Labels

Label	Type	Addr
main	.text	0
next	.text	3

Registers

Register	Value
$pc	0

Assembly

```
        .data
        .text
main:
        nop
        j next
        nop
next:
        nop
        j main
```

Design of Virtual Machine

1. MIPS registers (0-31, `pc`, `hi`, and `lo`)
    - ???
2. Memory for .data (a sequence of bytes, 512 bytes)
    - ???
3. Association between labels and memory locations or instructions
    - ???
4. The program, as a sequence of insturctions
    - ???

Design Patterns and Idioms

- Today we will discuss the use of design patterns and common idioms used to write canonical C++ code.
- Common C++ idioms
- Example: RAII
- Example: Copy/Swap
- Example: COW
- Design Patterns
- Example: Iterator

Common C++ Idioms

- All programming languages are equivalent in the sense that they are Turing complete.
- However, programming languages (or more properly the community of programmers) develop *idioms*, common ways of expressing ideas that leverages the semantics of that language.

Common C++ Idioms

- Simple example in C++: removing excess storage from a container, (e.g. a std::vector)
  - std::vector::shrink_to_fit

```
// Prior to C++11
std::vector<int>(c).swap(c);

//With C++11 (technically it is still a “non-binding request”)
c.shrink_to_fit();
```

Another simple C++ idiom: erase-remove

- What does the following print?

```
std::list<int> mylist;
mylist.push_back(0);
mylist.push_back(12);
mylist.push_back(31);
std::cout << mylist.size() << std::endl;

std::remove(mylist.begin(), mylist.end(), 12);
std::cout << mylist.size() << std::endl;
```

Another simple C++ idiom: erase-remove

- Remove actually does not actually remove! To really remove you use the "erase-remove" idiom.

```
mylist.erase(std::remove(mylist.begin(), mylist.end(), 12), mylist.end());
```

- See example code: `shrink.cpp`

Another simple C++ idiom: erase-remove

- Erase–remove idiom
- std::remove
- std::vector::erase

Example: RAII

- RAII stands for Resource Acquisition Is Initialization.
    - Resource acquisition is initialization
    - RAII
- See example code: `raii.cpp`

Example: Copy/Swap

- We can remove the code duplication and the self assignment test in the copy-assignment operator using the copy-swap idiom.
    - Copy-and-Swap Idiom in C++
    - Copy-and-swap
- See example code: `copyswap.cpp`

Example: Move semantics in C++11

- C++11 defines *move semantics* that add to RAII and the copy-swap idiom
    - Move semantics
    - std::move
- See example code: `copyswap11.cpp`

Example: Copy-on-Write (COW)

- A big difference between most `std::string` implementations and `QString` is the latter uses COW.
- COW is an optimization that lets objects share the same data as long as neither tries to change it, at which time a copy is made.
- Note: Matlab uses this for Matrices.
- COW has problems with concurrency, as we will see in a couple of weeks.
- See example code: `cow.cpp`

Design Patterns

- Design patterns are similar to Idioms but are less language specific.
- They are patterns in the sense of higher-order abstractions of code design.
- See the book Design Patterns: Elements of Reusable Object-Oriented Software
- There are many online compendium of patterns.

Example Design Pattern: PIMPL: Pointer-to-Implementation

- Pimpl decouples the definition and implementation of a class stronger than via private and public.
  - Pointer To Implementation
  - PImpl
- Can be usefull for abstracting platform differences without headers full of macros.
- Qt uses the Pimpl pattern extensively.
- See example code: `pimpl/*`

Example Design Pattern: Iterators

- Iterators are used throughout the standard library for accessing and manipulating containers.
- They are an abstraction of pointers.
- See example code: `iterators/*`

Criticisms of Design Patterns

- To some extent the patterns are ways of expressing things not naturally found in the language.
- Some people consider this a limitation of the programming language in question.
- It is easy to go overboard. Some patterns are overused (in my opinion), Singleton for example.

Next Actions and Reminders

- Read about the Factory and Model-View Pattern