ISA & CSE 141 Project

Slides by Hung-Wei Tseng
From Code to Running Programs

C program

Assembly Language

compiler

Object (Machine Code)

Assembler

Library (Machine Code)

linker

Executable (Machine Code)

Your Project!

Memory

loader
From Code to Running Programs

Your Project!

int main()
{
    printf("hello, world\n");
    return 0;
}

ldah gp,8192(t12)
lda gp,28464(gp)
ida sp,-16(sp)
stq ra,0(sp)
stq fp,8(sp)
mov sp,fp
ldah a0,-1(gp)
lda a0,32648(a0)
ldq t12,-32584(gp)
jsr ra,(t12),120006b40
ldah gp,8192(ra)
lda gp,28424(gp)
clr v0
br 120001198
mov fp,sp
ldq ra,0(sp)
ldq fp,8(sp)
ida sp,16(sp)
ret zero,(ra),0x1
nop

Your Project!
Assembly Language

- Text representation of machine instructions

Sample:

```
.text
la $1, table0 // load the address of a label table0 (pseudo instruction)
lw $2, $1 // $2 <= 0x000C0FFEE
lw $3, table0 // load the value at label table0 (pseudo instruction), $3 <= 0x000C0FFEE
lw $4, 3(table0) // $4 <= table1
lw $5, 1($4) // $5 <= 0x1DEADBEEF
sw $5, 1($1) // 0x001COFFEE is overwritten to 0x1DEADBEEF
li $6, 0xC0FFEE // load immediate (pseudo instruction), $6 <= 0xC0FFEE

.data
table0:
.word 0x000C0FFEE
.word 0x001C0FFEE
.word 0x002C0FFEE
.word table1
.table1:
.word 0x0DEADBEEF
.word 0x1DEADBEEF
.word 0x2DEADBEEF
.word 0x3DEADBEEF
.word 0x0DEADBEEF, 0x1DEADBEEF, 0x2DEADBEEF
.fill 10 0x0
```
Assembly Language

- Sections
  - Text Section: Instructions of the program
  - Data Section: Initialized data storage
- Keyword
  - Indicate the change of sections, how to deal with the data or ...
- Label
  - A name for an instruction/data address
- Pseudo Instruction
  - The shortcut to a sequence of instructions
Assembler

● Translate assembly code to machine code
  ○ From add $s0, $a1, $t7 to 0x00AF8020

● Develop an assembler
  ○ You may use the java-based Assembler framework
    ■ Accept your assembly code as input
    ■ Output
      ■ [$prefix]_i.coe: machine code
      ■ [$prefix]_d.coe: initial data memory
Tips of Using Framework

- Extend the Assembler class
  - AssembleCode() contains a two parse algorithm
    - parse #1: scan the code and labels
    - parse #2: replace labels with offsets (or addresses) and generate code
- Implement virtual methods:
  - processLabel, generateCode, updateProgramCounter, initialization, replaceInstructionLabel, and replaceMemoryLabel
AssembleCode() 1st Parse

Initialization()

getNextInputLine()

Is it Keyword?

Is it Label?

Is it an Instruction?

EOF?

processData(): store data section into “memory”.
- processAdditionalKeywords()

Is it an Instruction? yes

processLabel(): store in a table?
- processInstruction(): parse instruction into Instruction object
- updateProgramCounter(): update PC, be careful about pseduo inst.

*You need to implement all the methods in red
AssembleCode() 2nd Parse

- Calculate Label Address for PC
- Generate Code
- Calculate Label Addresses for Data
- Output Data

**Replace Instruction Label()**: if there is any label within the instruction, replace it with memory address.

**Generate Code()**: Generate the machine code according to the instruction you feed in!

**Replace Memory Label()**: Scan the “memory” object, replace all the labels with addresses.

*You need to implement all the methods in red.*
Simulator

- Simulate the behavior of a processor running your own ISA
  - Input: the assembled machine code
  - Output: what the code supposed to do
Simulator

- Develop a simulator
  - You may use the provided simulator framework
  - Tip:
    - Starting by modify the switch statement in the "execute" method in the provided ISASimulator class
Benchmarks

- Programs used to validate the functionality and performance of a computer system
  - Fibonacci numbers
    - Function calls
  - SuperGarbage
    - A virtual machine
    - Be careful of branch instructions.