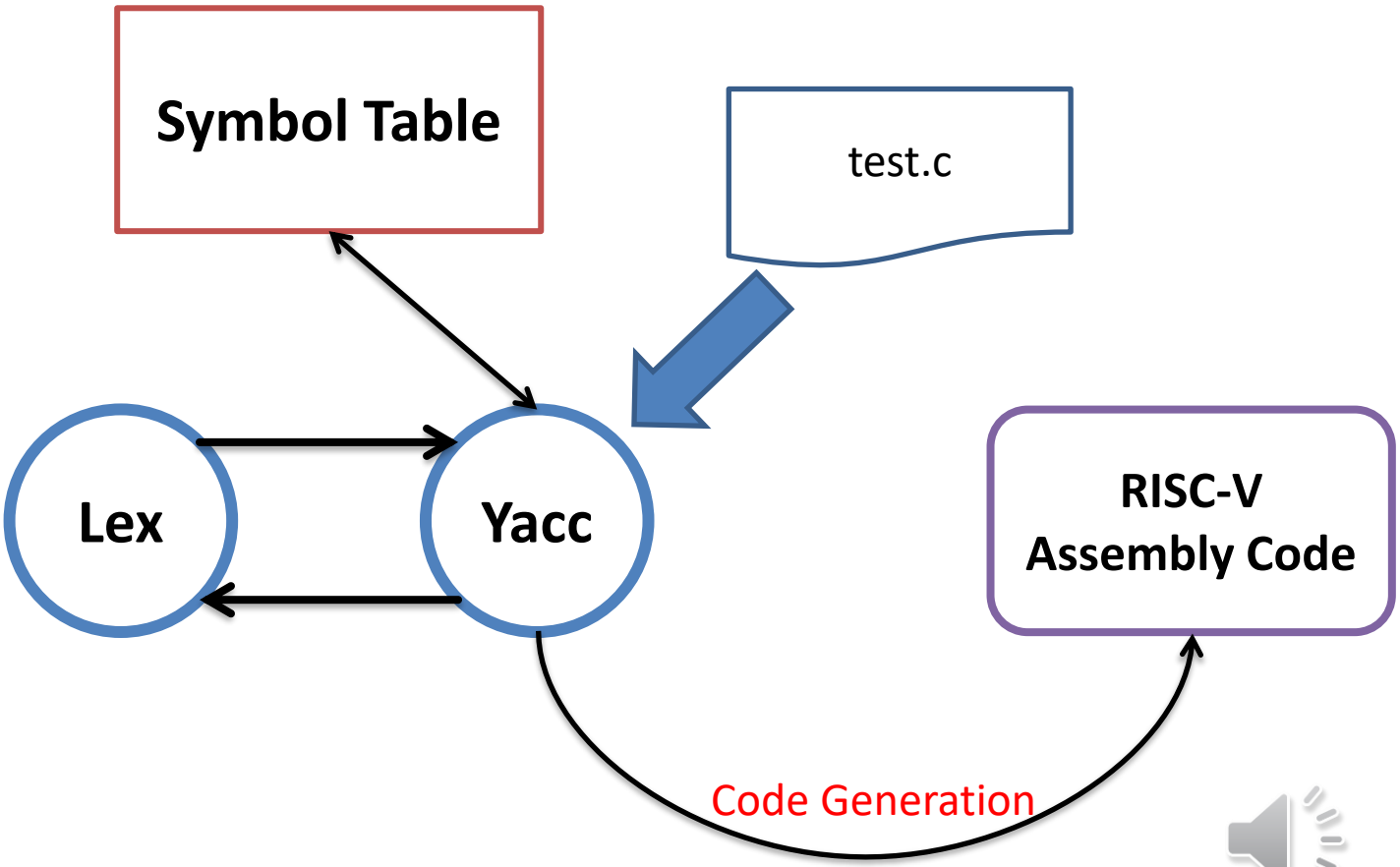
CS340400 Compiler Design
Homework 3

Submission Deadline: 2024/06/20 08:00

Demo Time: 2024/06/20~21

HW3 Architecture



Hints on Implementation

- Symbol Table
- Stack
- Generate Assembly Code

Symbol Table

- A table which keeps the information of symbols
  - E.g. scope, type, memory location, parameters, …
- When a symbol (variable/function) declaration is encountered, store the information of the symbol into the symbol table
- When a symbol is accessed later on, read the symbol table to find out how to access the symbol

Stack

- A process has a stack memory allocated for it
- Program can allocate memory on stack for variables
- Program can save temporary results on stack

Generate Assembly Code

```
/* parser.y */
/* this is just an example, and it's incomplete */
/* but it does provide a hint on how to use stack */
...
expr: expr '+' expr {
  ...
  fprintf(f_asm, "  lw t0, %d(sp)\n", $1->stack_offset);
  fprintf(f_asm, "  lw t1, %d(sp)\n", $3->stack_offset);
  fprintf(f_asm, "  add t0, t0, t1\n");
  ...
};
...
```

Note on Generating Code for HW

- We use 32-bit RISC-V, which means you should use `lw` and `sw` with 4-byte memory space to load/save registers

Execute Assembly on Andes Corvette-F1

Prerequisite

- Setup Arduino environment for Andes Corvette-F1 following another guide
- Have a piece of assembly code to execute
- Have access to the "assembly" (Arduino version) sample project provided by TAs

Execute Assembly Code

1. Replace content of "codegen.S" with your assembly code
   - Your codegen binary should be able to produce the whole content of "codegen.S" without any manual tweak
2. Click "Upload" in the "Sketch" menu in Arduino IDE

What Happens in the Background

1. There's a function declared with the signature `` `extern "C" void codegen() asm ("codegen");` `` in "assembly.ino"
   - It basically says that there's this external function that takes no argument and returns nothing with the assembly symbol of `` `codegen` ``
2. Arduino sees "codegen.S" in the project folder, builds it with an assembler, and links it to the final binary
   - The `` `.global codegen` `` directive exports the `` `codegen` `` symbol to the global scope, so it can be found by the linker

NTHU PL LAB

Execute Assembly on RISC-V Qemu Simulator

Prerequisite

- Have access to “Qemu" on TA's server
- Have a piece of assembly code to execute
- Have access to the "assembly" (Qemu version) sample project provided by TAs

Execute Assembly Code

1. Replace content of "codegen.S" with your assembly code
   - Your codegen binary should be able to produce the whole content of "codegen.S" without any manual tweak
2. Compile your program with `riscv32-unknown-elf-gcc main.c codegen.S`
3. Execute your program with `qemu-riscv32 a.out`

What Happens in the Background

1. In main.c, `void delay(int)` and `void digitalWrite(int, int)`, so you can use them in your "codegen.S"
2. In main.c, `void codegen()` is declared but not defined. Your "codegen.S" should define it
3. In main.c, the main function calls `void codegen()`, which should be your generated program

HW3 Specification

Format Rule

- At least, a function `` `void codegen();` `` would be present in the input file
  - Both the declaration and definition
  - This is the entry point of the generated program
- For each function, your codegen binary should be able to compile it into assembly codes with the following format:

```
.global <func_name>
<func_name>:
  (assembly implementation)
```

Special Function

- Function invocation codegen is not mandatory, except for `` `digitalWrite` `` and `` `delay` ``
  - Students can either implement generic function invocation codegen, or tackle these 2 functions specifically with Lex and Yacc modification

Special Function (cont.)

- `digitalWrite(pin, value)`
  - `pin`: an integer
  - `value`: `HIGH` or `LOW`
    - `HIGH` is `1`; `LOW` is `0`
  - Writes a HIGH/LOW signal to the specified pin
  - https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

Special Functions (cont.)

- `` `delay(ms)` ``
  - `` `ms` ``: an integer
  - Sleep for the specified time
  - https://www.arduino.cc/reference/en/language/functions/time/delay/

Special Functions (cont.)

- Prepare arguments in `a0`, `a1`, …
- Call `jal ra, <func_name>` to jump to function and set return address in `ra`
- E.g. to invoke `delay(1000);`:

```
li a0, 1000
jal ra, delay
```

- Note that you should save caller-saved registers onto stack before invocation and restore them afterwards
  - RISC-V Calling Convention Reference: <https://riscv.org/wp-content/uploads/2015/01/riscv-calling.pdf>

Grading Policies

Misc. Rule

- Come to the demo session, or get 0
  - Still true for those who can't pass at least 1 testcase
- No static/manual optimization. We want to see how your codegen binary handles the program structure
  - E.g. no constant propagation, no constant folding, no dead code elimination, …
- Token/Syntax requirement is the same as HW1/2

Testcase Rule

- Testcases are levelled. Each level has several testcases, but only 1 of them (chosen by the TA at demo time) will be tested in a demo session to prove that the tested codegen binary passes the level
- All testcases are public, so feel free to test it by yourself

Demo Rule

- On 2024/06/20 (Thurs.) ~21 (Fri.) at資電館 844
- Tell us what grade level you want to demo
- Show that your codegen binary can compile the testcase correctly to get the score
- You'll be asked about how your codegen binary handles certain structures of the testcase
- simulator demos: conducted on TA's server with command line
- board demos: conducted on your own computer

Testcase "Basic"

- Grade Lv.: E
- #testcases: 1
- Description
  - Your compiler should handle the basic structures of testcases correctly, including but not limited to `` `codegen` `` function declaration/definition, and invocations of special functions

Testcase "Arithmetic Expression"

- Grade Lv.: D
- #testcases: 2
- Description
  - Your compiler should handle arithmetic expressions composed of `+`, `-`, `*`, `/` and parentheses correctly

Testcase "Pointer"

- Grade Lv.: C
- #testcases: 2
- Description
  - Your compiler should handle single-level pointers correctly

Testcase "Jump"

- Grade Lv.: B
- #testcases: 2
- Description
  - Your compiler should handle branching/loop statements correctly
  - Your compiler should handle 1D array correctly

Testcase "Function"

- Grade Lv.: A
- #testcases: 2
- Description
  - Your compiler should handle generic function invocations with the RISC-V calling convention correctly
  - RISC-V Calling Convention Reference: <https://riscv.org/wp-content/uploads/2015/01/riscv-calling.pdf>

Grading

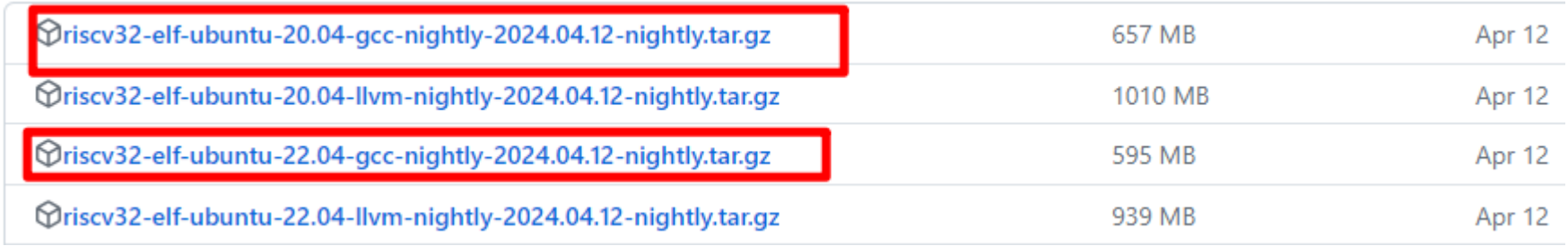
- Question on demo: 10 pts
- demo: 90 pts(max)
  - Testcase E: get 50 pts
  - Testcase D: get 60 pts
  - Testcase C: get 70 pts
  - Testcase B: get 80 pts
  - Testcase A: get 90 pts

Submission

- Server: Source code
  - Upload to eeclass
  - Upload files:
    - scanner.l
    - parser.y
    - A `makefile` to compile your code
      - The output of the makefile must be named `codegen` and marked as executable
    - All other files needed to compile your `codegen`

Environment (qemu)

1. Right click to get the link
   - choose by OS version
   - Riscv toolchain github release
2. Execute "$ wget <link>"



If you want to run on your computer

Environment (qemu)

3.Execute “$ tar -zxvf ***.tar.gz”

4.Executables in./riscv/bin