THE UNIVERSITY OF ARIZONA
DEPARTMENT OF COMPUTER SCIENCE

# CSc 453: Spring 2024

## Assignment 4 milestone 3

Start: April 19, 2024
Due: 11:59 PM April 29, 2024

## General

This assignment involves final code generation (i.e., MIPS assembly) for the full C-- language. An example three-address instruction set is given here; details of its translation to MIPS assembly code are given here.

## Required functionality

### *1. Programming requirements*

#### 1.1. General

Your program should extend the functionality of your compiler from Assignment 4 milestone 2 to generate MIPS assembly code for the entire C-- language:

- Functionality from earlier assignments, i.e., syntax and semantic checking as well as AST construction, should be retained.
- Additionally, when the command-line argument `--gen_code` is specified, your program should:
    - perform additional processing to handle a special function, **`println(`***expr***`)`**, that we will use for print out the value of the expression *expr* (see the appendix at the end of this document);
    - for each function *f* in the input program:
        - traverse the AST for *f* to generate a list of three-address instructions at each AST node; and
        - traverse the three-address code at the root of the AST for *f* and emit MIPS assembly code. The MIPS code should be written to stdout.

#### 1.2. Makefile

You should submit a Makefile that provides (at least) the following targets:

**make clean**:
Deletes any object files ( *.o ) as well as the file 'compile'

**make compile**:
Compiles all the files from scratch and creates an executable file named 'compile'.

### *2. Driver code and program invocation*

The driver file driver.c is the same as for Assignment 4 Milestone 2. It recognizes the command-line argument `--gen_code` used for this assignment and sets the variable `gen_code_flag` to 1 if it is invoked with the argument `--gen_code`.

## Interface requirements

Your program should be written to interface appropriately with the driver code mentioned above. To this end, your code should provide the following:

```
int parse();  /* the parser function */
```

The driver code will set the value of an int variable `gen_code_flag` based on whether or not a command-line argument `--gen_code` is specified (see the code in the file **driver.c** linked above). The parser should print out MIPS assembly code if and only if `gen_code_flag` has a non-zero value.

## Three-address and final code generation

An example three-address instruction set is given here; details of its translation to MIPS assembly code are given here.

## Files you need to submit

- Makefile that supports the functionality described above;
- any additional files implementing your scanner and parser.

---

## Appendix. The println() function

We will use a special function, **`println()`**, to print out values computed when the MIPS code generated by your compiler is executed. This function takes a single integer value as argument and print out that value followed by a newline, in effect behaving as follows:

```
void println(int x) {
    printf("%d\n", x);
}
```

The examples below show the output generated when the code generated from the input program is executed on SPIM:

| Example 1 | Example 2 |
|---|---|
| Input program:<br><br>`int main() {`<br>`  println(34567);`<br>`}` | Input program:<br><br>`int main() {`<br>`  int x;`<br>`  x = 12345;`<br>`  println(x);`<br>`}` |
| SPIM output:<br><br>SPIM Version 8.0 of January 8, 2010<br>Copyright 1990-2010, James R. Larus.<br>All Rights Reserved.<br>See the file README for a full copyright notice.<br>Loaded: /usr/lib/spim/exceptions.s<br>**34567** | SPIM output:<br><br>SPIM Version 8.0 of January 8, 2010<br>Copyright 1990-2010, James R. Larus.<br>All Rights Reserved.<br>See the file README for a full copyright notice.<br>Loaded: /usr/lib/spim/exceptions.s<br>**12345** |

Note that the input program *uses* println() but does not *define* it. This is similar to the way we use library functions like printf(). To make this work, your compiler must do the following:

1. Before parsing: Initialize the symbol table appropriately so that calls to println() do not cause an "undeclared identifier" error during semantic checking. (This is conceptually analogous to processing a prototype for printf() from the file stdio.h).

2. During code generation: generate the following sequence of MIPS instructions for println(). (This is conceptually analogous to linking in the library code for printf() statically).

```
.align 2
.data
_nl: .asciiz "\n"

.align 2
.text
# println: print out an integer followed by a newline
```

```
_println:
   li $v0, 1
   lw $a0, 0($sp)
   syscall
   li $v0, 4
   la $a0, _nl
   syscall
   jr $ra
```