Assembly

Compiler Construction

2024-12-10

Prof. Dr. Michael Kuhn
michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

- Assembly is the next step on the way to a working compiler
- x86 or ARM assembly can be quite complex and complicated
  - A relatively small subset should suffice for a minimalistic compiler
- We will mostly use GCC assembly for x86

Assembly...

- Directives: Begin with a dot and indicate structural information
  - Example: `.global main`
- Labels: End with a colon and denote special positions in the code
  - Example: `main:`
- Instructions: Actual assembly code to be executed
  - Example: `JMP main`

Outline

Group Exercise

- Come up with an assembly version of the following for loop.

```
1 main: function void () = {
2     i: integer;
3     j: integer = 0;
4     for (i = 0; i < 42; i++) {
5         j++;
6     }
7 }
```

Group Exercise

- Come up with an assembly version of the following for loop.

```
1  main:
2      ...
3      MOVL $0, %ebx # j
4      MOVL $0, %eax # i
5      JMP .for_check
6  .for_body:
7      ADDL $1, %ebx
8      ADDL $1, %eax
9  .for_check:
10     CMPL $41, %eax
11     JLE .for_body
12     ...
```

Group Exercise

- Come up with an assembly version of the following switch statement.

```
1  void foo (int i) {
2      switch (i) {
3          case 0:
4              i++;
5              break;
6          case 1:
7              i--;
8          case 2:
9              i -= i;
10     }
11 }
```

Group Exercise

- Come up with an assembly version of the following switch statement.

```
foo:
    ...
    # Assume i in %eax
    CMPL $0, %eax
    JE .case_0
    CMPL $1, %eax
    JE .case_1
    CMPL $2, %eax
    JE .case_2
    JMP .end
.case_0:
    ADDL $1, %eax
    JMP .end
.case_1:
    SUBL $1, %eax
.case_2:
    MOVL $0, %eax
.end:
    ...
```

Group Exercise

- Come up with an assembly version of `alloca`. Does it need any special handling?

```
1 void foo (void) {
2     void* bar = alloca(128);
3 }
```

Group Exercise

- Come up with an assembly version of `alloca`. Does it need any special handling?

```
1 foo:
2     ...
3     SUBQ $128, %rsp
4     MOVQ %rsp, bar
5     ...
```

- `alloca` needs to be handled as a compiler built-in

Group Exercise

- What could go wrong in the following code snippet?

```
1  void foo (void) {
2      void* bar;
3      ...
4      bar = alloca(1024);
5      ...
6  }
7  int main (void) {
8      for (int i = 0; i < 100000; i++) {
9          foo();
10     }
11 }
```

Group Exercise

- What could go wrong in the following code snippet?

```
1  void foo (void) {
2      void* bar;
3      ...
4      bar = alloca(1024);
5      ...
6  }
7  int main (void) {
8      for (int i = 0; i < 100000; i++) {
9          foo();
10     }
11 }
```

- Inlining could lead to a stack overflow

Outline

Summary

- Assembly is CPU-specific
  - x86 and ARM support different instructions
- There are different assembly dialects
  - Intel lists the source first, while AT&T/GCC lists the destination first
- Different instruction suffixes and registers support different data sizes
  - Several addressing modes and additional registers give more flexibility

References

[Thain, 2020] Thain, D. (2020). ***Introduction to Compilers and Language Design: Second Edition.*** http://compilerbook.org/.