CSED211: Lab. 5

Optimization

조승혁
shhj1998@postech.ac.kr
POSTECH

2023.11.6

Table of Contents

GCC

- GCC provides some optimization options.
  - https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
  - The optimization option tends to **improve the performance / reduce the size** of the binary.
- We can enable the optimization options by using the `-O{level}` argument.
  - `-O, -O1, -O2, -O3, -O0, -Os, -Ofast, …`
  - `level` controls the optimization level.
  - Higher optimization level increases the compilation time.

GCC Optimization Levels (1)

- `-O0` (Default)
  - Reduces compilation time.
  - Makes debugging produce the expected results.
- `-O1, -O`
  - Consume more time and memory for compilation.
  - Reduces the code size and execution time.

GCC Optimization Levels (2)

- `-O2`
  - Performs nearly all supported optimizations.
  - Does not involve space-speed tradeoff.
  - Improve the performance with increased compilation time.
  - ...
- `-O3`
  - Turns on more optimization flags.
  - Does not guarantee better performance.

GCC Optimization Levels (3)

- `-Os`
  - Removed the optimizations that increases code size from `-O2`.
  - Tunes for code size rather than execution speed.
- `-Ofast`
  - Included the optimizations which disregard strict standards compliance to `-O3`.
- `-Og`
  - Optimizes debugging experience.
  - Only for edit-compile-debug cycle.
  - Recommended when debugging the program instead of `-O0`.

GCC Optimization Levels: Summary

Option	Optimization Level	Execution Time	Code Size	Memory Usage	Compile Time
`-O0`	Optimization for compilation time (default)	+	+	-	-
`-O1 / -O`	Optimization for code size and execution time	-	-	+	+
`-O2`	Optimization more for code size and execution time	--		+	++
`-O3`	Optimization more for code size and execution time	---		+	+++
`-Os`	Optimization for code size		--		++
`-Ofast`	`-O3` with fast non accurate math calculations	---		+	+++

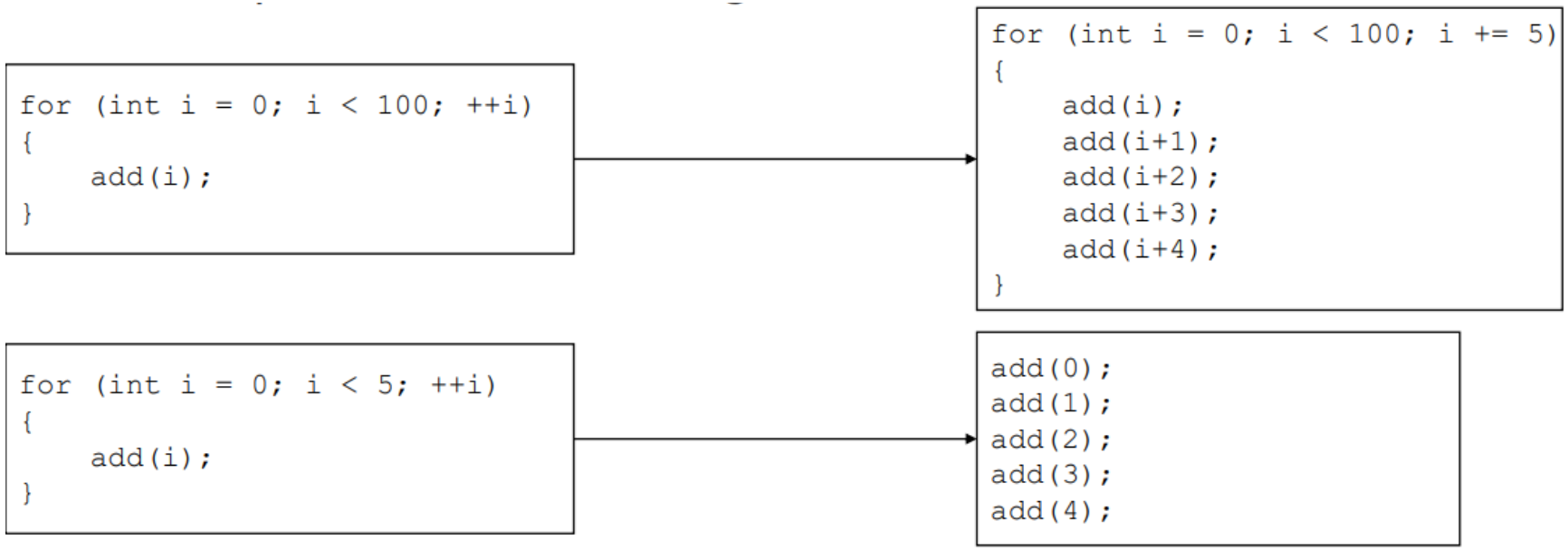
GCC Optimization Flags

- Optimization level includes optimization flags.
  - https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
- For example, `-O1` uses the following flags:

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion
-fif-conversion2
-finline-functions-called-once
-fipa-modref
-fipa-profile
-fipa-pure-const
-fipa-reference
-fipa-reference-addressable
-fmerge-constants
-fmove-loop-invariants
-fmove-loop-stores
-fomit-frame-pointer
-freorder-blocks
-fshrink-wrap
-fshrink-wrap-separate
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-pta
-ftree-scev-cprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-ter
-funit-at-a-time
```

GCC Optimization Flags - Example

- `-funroll-loops`
  - Unrolls loops whose number of iterations can be determined at compile time or upon entry to the loop.
  - Turns on complete loop peeling, i.e., complete removal of loops with a small constant number of iterations.
  - The code size becomes larger.



GCC Optimization - Example

- `gcc -O1 -fno-defer-pop -o test test.c`
  - Use optimization level `-O1` and optimization flag `-fno-defer-pop`.
  - The further flags override the flags from the optimization level.
  - i.e., `-O1` set `-fdefer-pop` as default.

Manual Optimization

- We can optimize the program by ourself!
  - Remove redundant variables, reduce the number of function calls, exploit locality, …
  - Or write the ASM code manually!

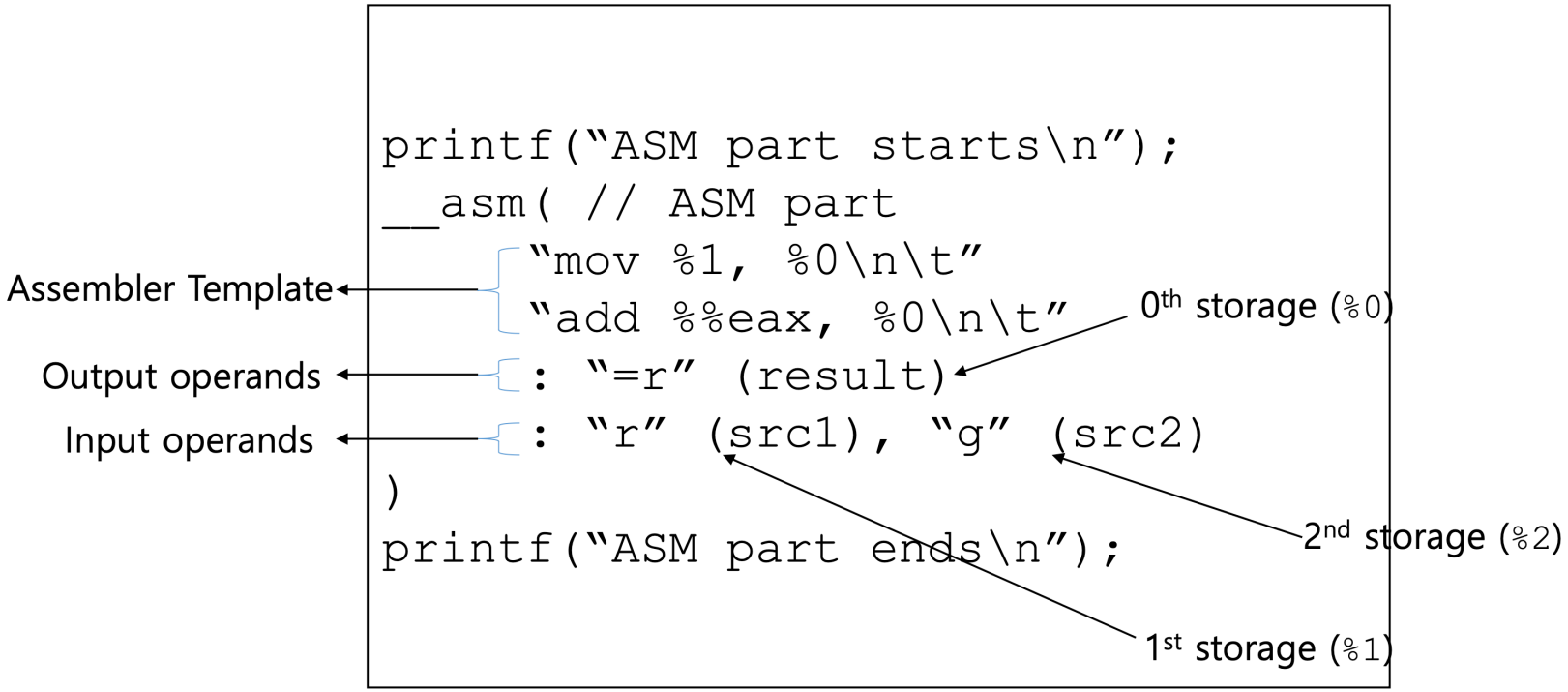
ASM Code in C (1)

- We use `__asm()` or `__asm__()` function for inline assembler.
- It can be used anywhere inside the C / C++ code.
- We use inline ASM for:
  - Spot-optimizing speed-critical sections of code.
  - Making direct hardware access for device drivers.

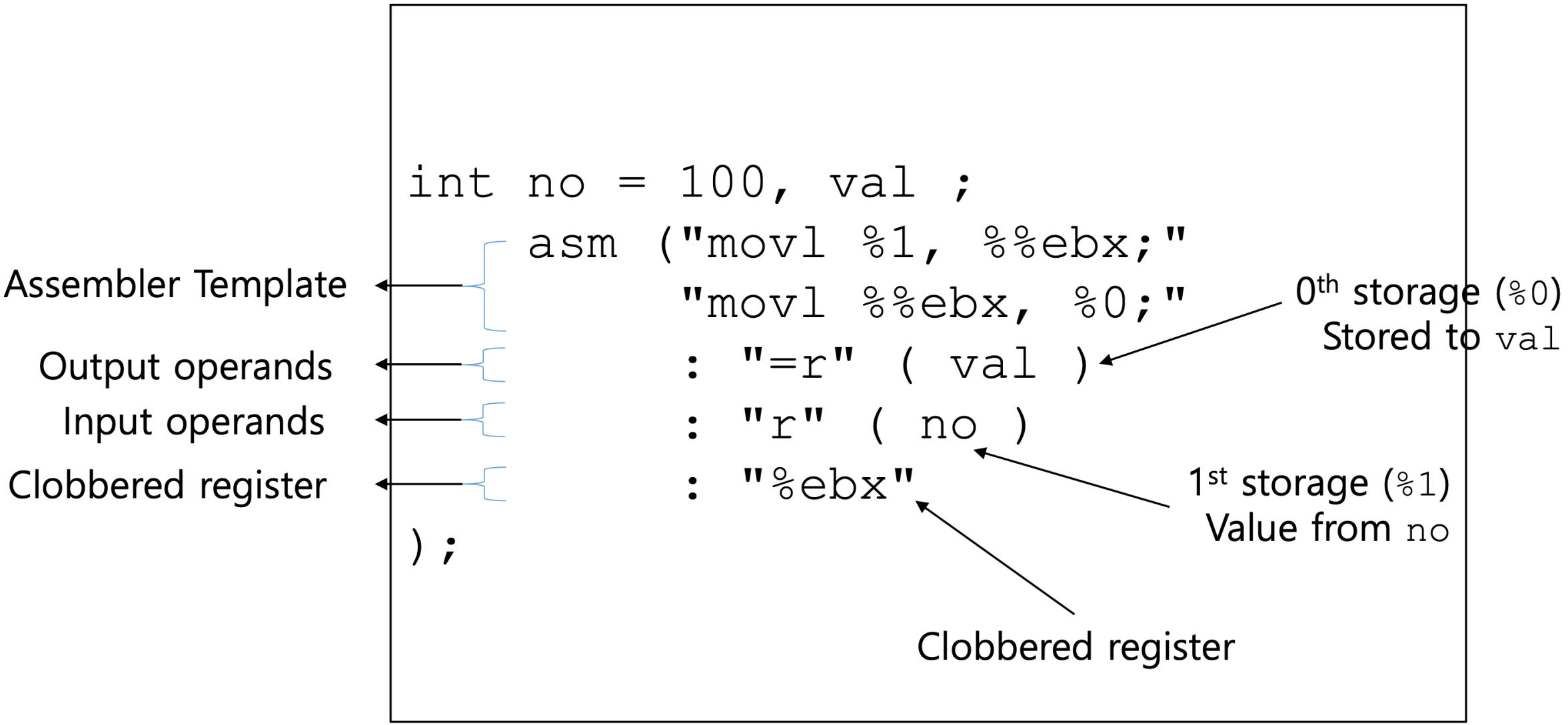
```
(c code)…
__asm( // ASM part
          …
          (ASM)
          …
          : OutputOperands
          : InputOperands
)
(c code)…
```

ASM Code in C (2)

- Assembler template is the literal string with assembler instructions.
  - We divide each line with the character `\n\t`.
- `%n`: register mapped to the `n` th argument.
- `%, {, |, }` should be used with `%`.



ASM Code in C (3)



Practice 1: GCC Optimization (1)

- We will test the following GCC optimization levels with the code `loops.c`.
  - `-O1, -Os, -O2.`

```c
#include<stdio.h>
#include<time.h>

int main()
{
        int i = 0;
        clock_t begin = clock();
        for (; i<1000000000; i++)
                if (i % 100000000 == 0)
                        printf("loading\n");
        clock_t end = clock();
        double spent_time = (double)(end - begin);
        printf("spent time: %f\n", spent_time);
        return 0;
}
```

Practice 1: GCC Optimization (2)

- Use `time` command to see the compilation time.
  - ex) `time gcc loops.c -O<level>`
- Run the binary file generated with gcc and check the spent time.
  - ex) `./a.out`

Practice 2: ASM Code in C

- Fill in the blank parts of `asm.c`.
  - The program should results `105`.
- Hints
  - `mov v1 v2` -> copy the value of `v1` to `v2`.
  - `add v1 v2` -> `v2 = v1 + v2`

```c
#include<stdio.h>

int main(){
        int no=100;
        int val;
        asm(
                /*
                 * Write an instruction that moves value of no to %ebx
                 * Write an instruction that adds 5 (constant) to %ebx
                 * Write an instruction that moves value of %ebx to return
                 */
                : "=r" (val)
                : "r" (no)
                : "%ebx"
        );

        printf("%d \n", val);
        return 0;
}
```

Quiz