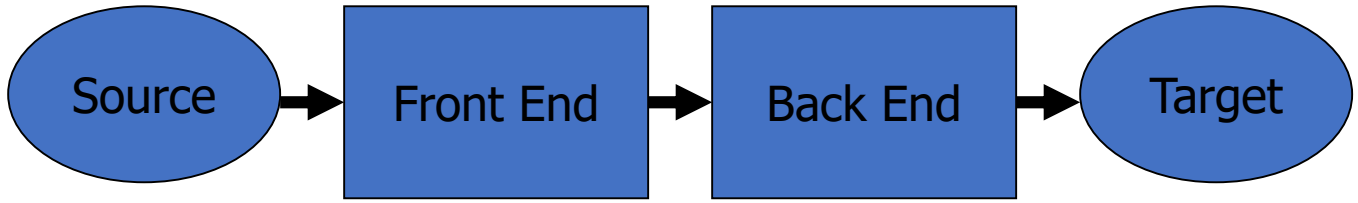
CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- Back End
- Preliminaries about HW1

Structure of a Compiler

- Still remember that? We are introducing Back End now!
- A very high level description
  - Front end: analysis
    - Read source program and understand its structure and meaning
  - Back end: synthesis
    - Generate equivalent target language program



Back End

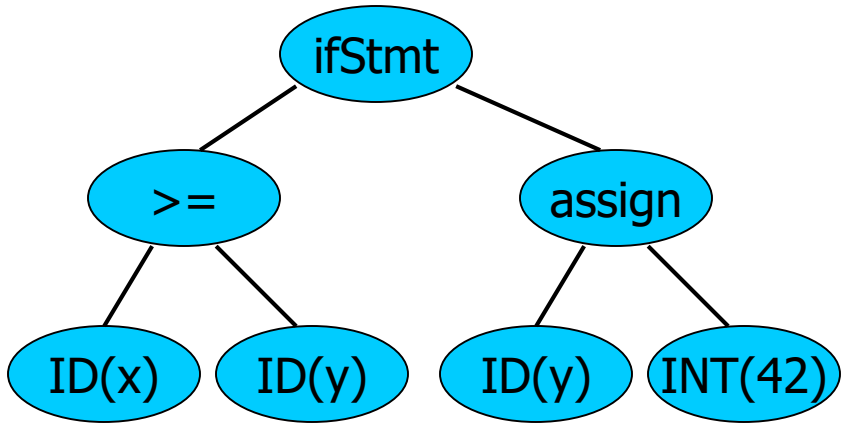
- Responsibilities (jobs for the Backend)
  - Translate IR into target machine code
  - Should produce “good” code
    - “good” = fast, compact, low power consumption (pick some)
  - Should use machine resources effectively
    - Registers
    - Instructions
    - Memory hierarchy

Back End Structure

- Typically split into two major parts with sub phases
  - Optimization – code improvements
  - Code generation
    - Instruction selection & scheduling
    - Register allocation

The Result

- Input
  - If (x >=y)
    y = 42;



The Result (Cont.)

- Output

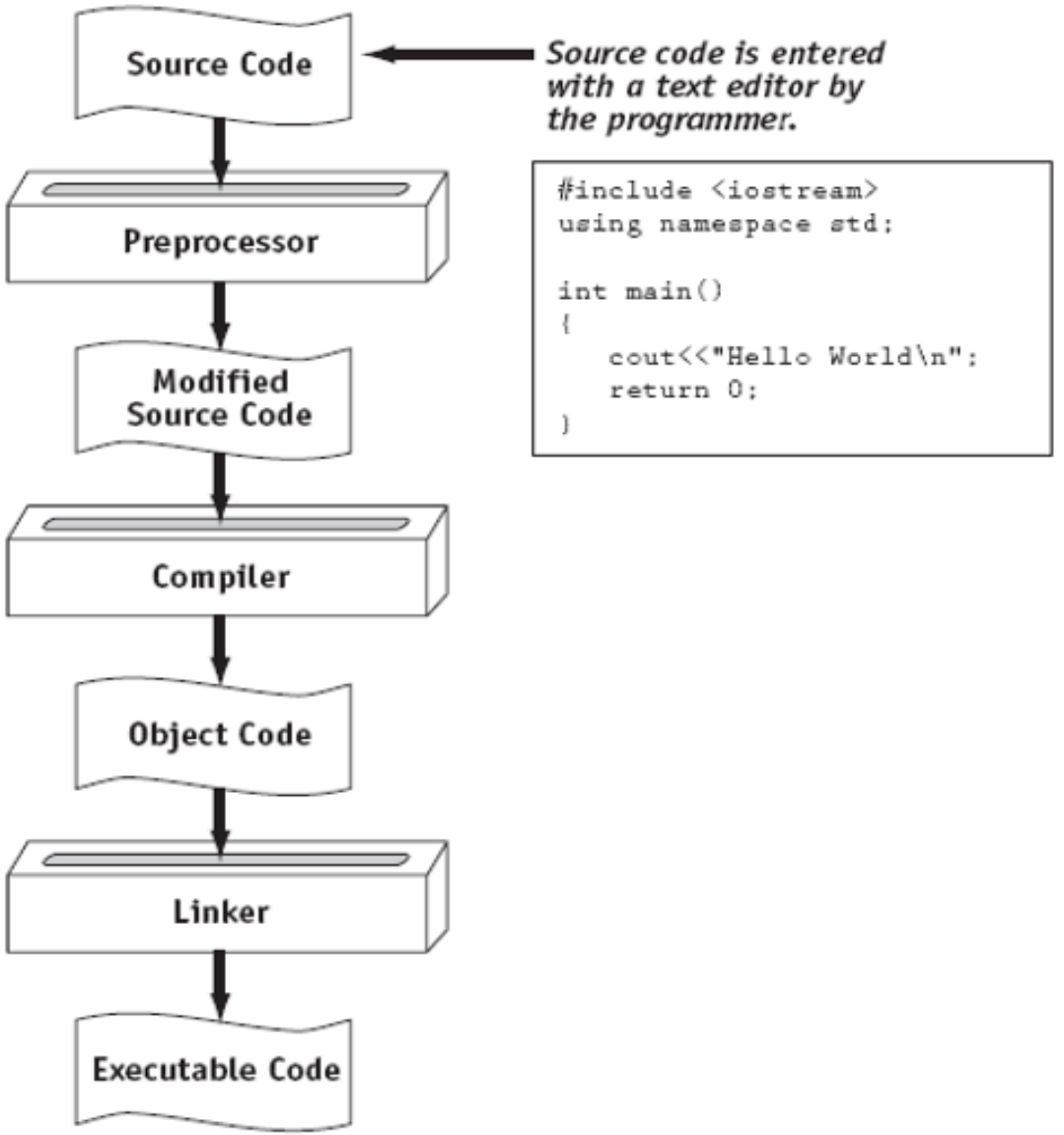
```
mov   eax,[ebp+16]
cmp   eax,[ebp-8]
jl        L17
mov    [ebp-8],42
L17
```

Preliminaries about HW1

- Forget about IR, AST, Lexical Analysis, Tokens!
- Get you hands dirty!
- In your HW1, you need to design & implement a compiler that can compile an integer from user's command line input! (easier version!)
  - The compiler is written in C language

Preliminaries about HW1

- What I'm going to do today, is to introduce a compiler that can accept a user's command line input.
- The compiler body is written in C language
- Supports "+" and "-" computation. (A more complicated version!)
- This is totally like a compiler can read in a source code file and output machine code (binary code)



Preliminaries about HW1

- Print out the Assembly “main:”
- Read the first single character and copy (mov) that to “rax” ← a register
- While there is still something…
- I keep reading the user’s input…
- If I meet a “+” sign ?
- I move onto the next character
- Use the “addition” (add) to add that content to “rax”
- Or, use the “subtraction” (sub) to sub that content to “rax” (do the same thing for sub)
- If I meet something “weird” character, I can complain and do some simple error handling
- When there is nothing in the user’s input, I can print the Assembly return (ret)

addsub.c

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

int main(int argc, char **argv) {
    if (argc != 2)
        fprintf(stderr, "Num. of args is wrong!");
        return 1;
    }

    char *p = argv[1];

    printf(".intel_syntax noprefix\n");
    printf(".global main\n");
    printf("main:\n");
    printf(" mov rax, %ld\n", strtol(p, &p, 10));

    while (*p) {
        if (*p == '+') {
            p++;
            printf(" add rax, %ld\n", strtol(p, &p, 10));
            continue;
        }

        if (*p == '-') {
            p++;
            printf(" sub rax, %ld\n", strtol(p, &p, 10));
            continue;
        }

        fprintf(stderr, "Unexcpected chars: '%c'\n", *p);
    }

    printf(" ret\n");
    return 0;

}

```

Preliminaries about HW1

- Please follow the tutorial I posed in Lecture 0 to setup your MinGW to support gcc, if you are using Windows system
- The “flow” is like this, I use 10+20-1 for our test input string!
  - Write the compiler source code in C, named addsub.c
  - gcc -o addsub addsub.c
  - ./addsub ’10+20-1’ > addsub.s
  - See the content of addsub.s and check if the assembly code is correctly printed from the addsub ?
  - gcc -o addsub.target addsub.s
  - ./addsub.target
  - echo $?
  - See if you can get 29 ?
  - (This is executed in x86-64 Linux environment)

Preliminaries about HW1

- Your job in HW1
  - The compiler can accept an integer input (or 2 integers as input, if it is necessary)
  - When the assembly code gets the integer, it can do some other applications (i.e. functions calls, if it is necessary)
  - I will post the “detail” of HW1, maybe this afternoon or tomorrow, in Blackboard.
  - Due date? One week. Because this is an easy one.