Project 4

Code Generation

Virtual Machine & Optimization Laboratory
Dept. of Electrical and Computer Engineering
Seoul National University

Projects

1. Lexical analyzer
2. Yacc programming
3. Semantic analysis
4. **Code generation**

Code generation & Stack Simulator

Input

Input C code (t.c)

```
int main() {
    write_string("hello world\n");
}
```

subc

Generated IR code (t.s)

Output

```
shift_sp 1
push_const EXIT
push_reg fp
push_reg sp
pop_reg fp
jump main
EXIT:
exit
main:
main_start:
Str0. string "hello world\n"
push_const Str0
write_string
main_exit:
push_reg fp
pop_reg sp
pop_reg fp
pop_reg pc
main_end:
Lglob. data 0
```

Test

Execution on stack simulator

```
smb1221@capella:~/compiler/sim$ ./sim t.s
code area size 12
data area size 14
hello world
program exits
```

Code Generation

스택 기반의 중간코드(IR)를 생성

- 자바 바이트코드와 유사함

생성된 코드를 스택 시뮬레이터에서 실행

코드는 subc.y의 Embedded action에서 생성

문법적으로 잘못된 코드(syntax, semantic)는 입력되지 않음

Stack Simulator

operand들을 스택에 push하고 pop해서 연산을 수행한 뒤, 결과를 다시 스택에 push하는 구조

- ex) JavaVM

instruction

- ex) add, sub, push_reg, pop_reg

설치 및 사용법

- 강의 홈페이지에서 다운받은 뒤, 압축을 풀고 make를 실행하면 sim파일 생성
- ./sim [file_name.s]
- 동봉된 test.s로 테스트해볼 수 있다.

Registers

SP

- 스택을 가리키는 포인터
- 주로, 지역 변수의 값을 접근하기 위해 사용

FP

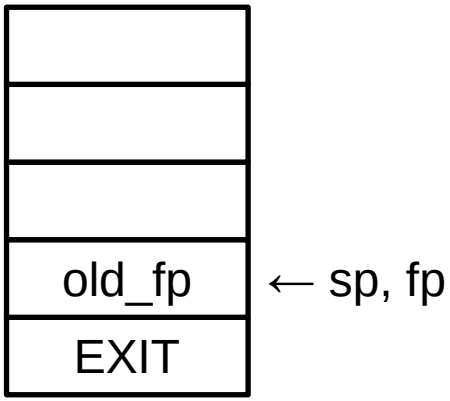
- 스택 프레임 포인터
- 함수의 호출, 리턴에 사용

PC

- 현재 수행중인 프로그램의 program counter
- branch를 수행하기 위해서는 PC값을 변경

Start up code

```
        shift_sp 1
        push_const EXIT
        push_reg fp
        push_reg sp
        pop_reg fp
        jump main
EXIT:
        exit
main:
```



calling convention에 따라 다소 변할 수 있음.

main함수에는 args가 없다고 가정

INSTRUCTION SET

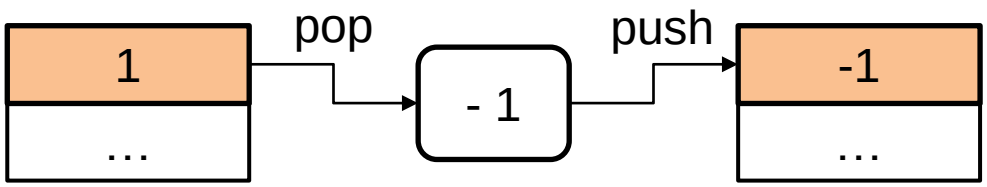
Arithmetic / Logic Instruction

Unary operation

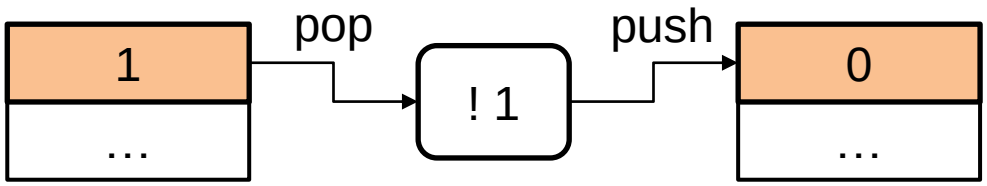
- pop top element of stack
- apply operation
- push result onto stack

Example

- negate



- not



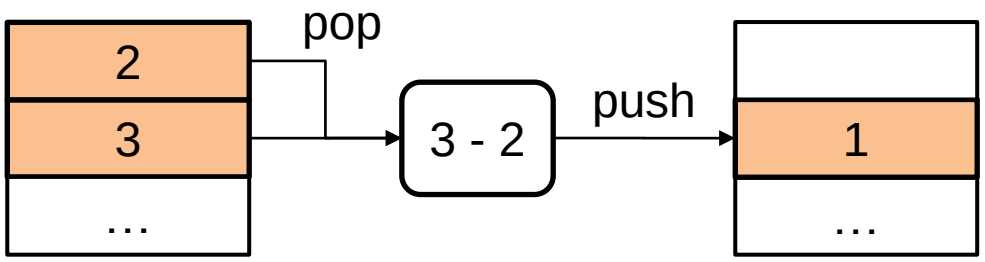
Arithmetic / Logic Instruction

Binary operation

- pop two top elements of stack
- apply operation as top element on right hand, second element of left hand
- push result onto stack

Example

- sub



Control Instruction

Terminate program

- exit
- 실행중인 프로그램을 무조건 종료

Unconditional jump

- jump [label] [+/- offset]
- example
  - jump L 6 → pc = L + 6
  - jump L → pc = L
  - jump 6 → pc = 6

Control Instruction

Conditional jump

- pop top element of stack
- branch_true [label] [+/- offset]
- branch_false [label] [+/- offset]

pop한 값이 1인 경우 지정된 위치로 점프하고 0인 경우는 다음 코드를 수행

Stack Manipulation Instruction

push

- push_const <constant>
- push_reg <reg>
- example
  - push_const Str0

  - push_reg fp

Stack Manipulation Instruction

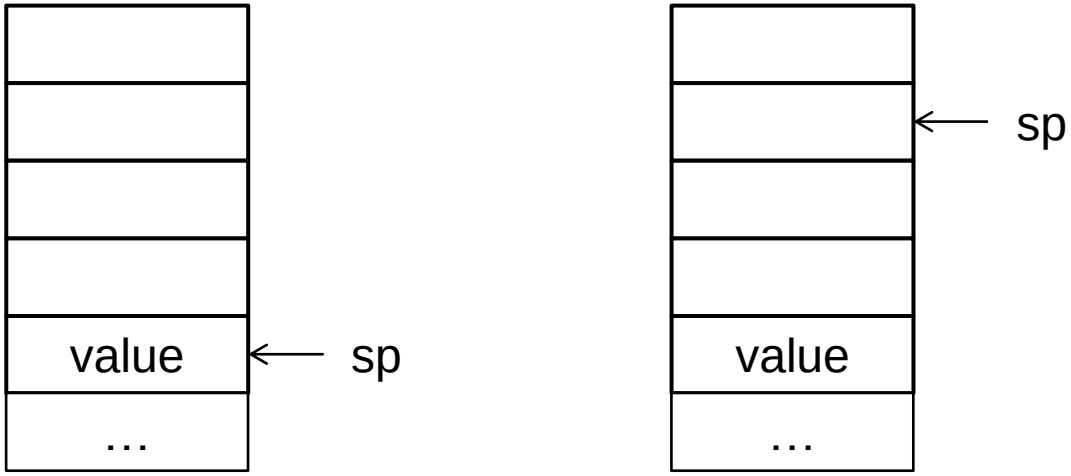
pop

- pop_reg <reg>
- example
  - pop_reg fp

Stack Manipulation Instruction

shift stack pointer

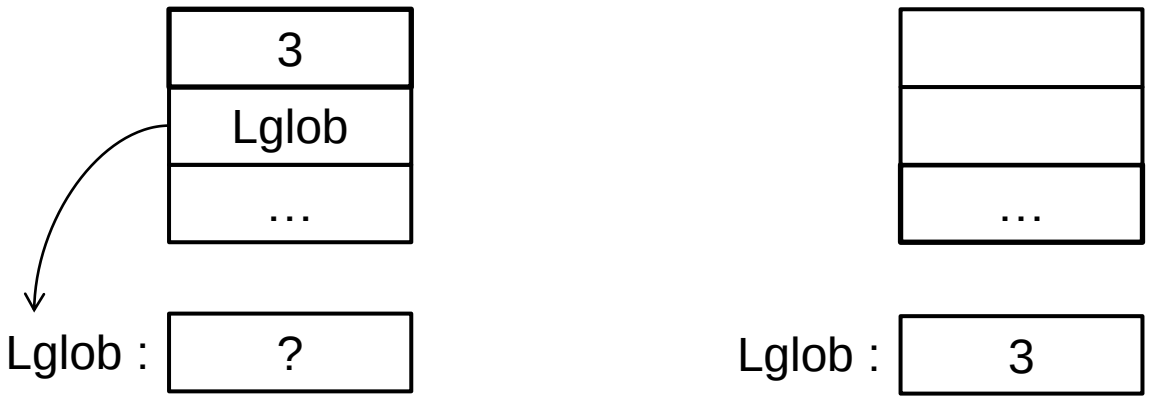
- shift_sp <integer constant>
- 지역 변수를 위한 스택 프레임 할당을 위해 사용
- example
  - shift_sp 3



Assign / Fetch Instruction

Assign value into specified address

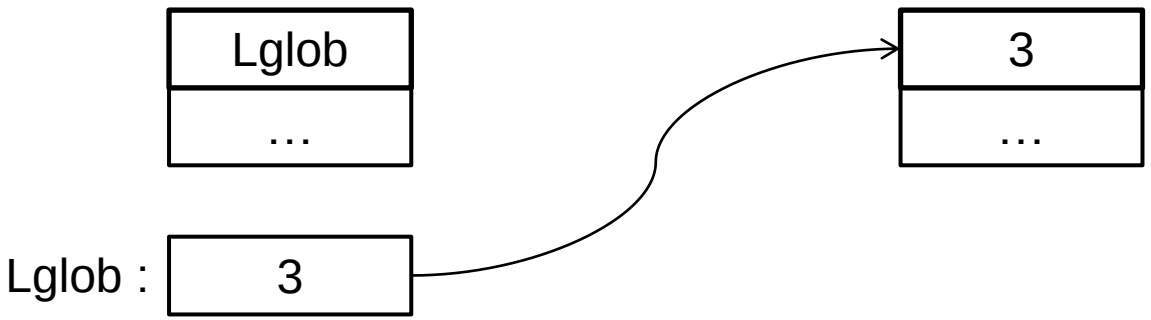
- example
  - push_const Lglob
  - push_const 3
  - assign



Assign / Fetch Instruction

Fetch value from specified address

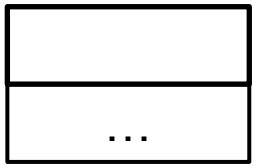
- example
  - push_const Lglob
  - fetch



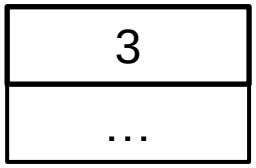
I / O Instruction

Input

- 숫자나 문자를 입력받고 스택에 push
- read_int: 숫자(integer)를 입력 받음
- read_char: 문자(character)를 입력 받음
- example
  - read_int



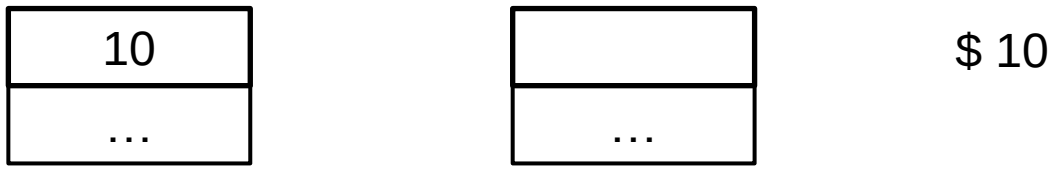
$ read int:
$ 3



I / O Instruction

Output

- POP한 뒤 화면에 숫자나 문자, 문자열을 출력한다
- write_int: 화면에 숫자를 출력
- write_char: 화면에 문자를 출력
- write_string: 화면에 문자열을 출력
- example
  - write_int



I / O - TODO

입력이 되는 C코드에서 read, write 함수를 지원

- 프로그램이 정확히 동작하는 지를 체크

C코드에서 I/O함수를 찾으면 해당하는 I/O instruction을 생성하도록 구현

example

```
int main() {
        int i;
        i = 5;
        write_int(i);
}
```

* read 함수는 Instruction set에 있긴 하지만,
본 프로젝트에 쓰이지 않을 것이므로 구현하지 않아도 됨

More Details on Stack Machine

스택 머신은 asm.l과 gram.y 파일로 작성

gram.y

- 각 instruction의 실제 동작은 simulate_stack_machine 함수에서 찾을 수 있음
- 코드, 스택 및 데이터영역의 크기와 오프셋을 확인할 수 있음

IMPLEMENTATIONS

Global Variables

전역 변수 저장공간을 설정

- <label>. data <size>
- size는 word 단위
- int, char, pointer 모두 1 word로 생각한다

```
int global_1;      1
int global_2;      1

struct _str1{
    int x;         1
    int y;         1
    struct _st2{
        int z;     1
        int w[5];  5
    } strstr
} sample_str;
    …
```

```
…
main_final:
            push_reg sp
            pop_reg sp
            pop_reg fp
            pop_reg pc
main_end:
Lglob. data 10
```

Global Variables

전역 변수에 값 대입하기

```
push_const Lglob+4    전역 변수 시작주소 + 오프셋
push_const 12                대입하고 싶은 값
assign                       대입
```

전역 변수에서 값 가져오기

- 불러 온, 전역변수의 값은 스택에 저장됨

```
push_const Lglob+4    전역 변수 시작주소 + 오프셋
fetch                        값 가져오기
```

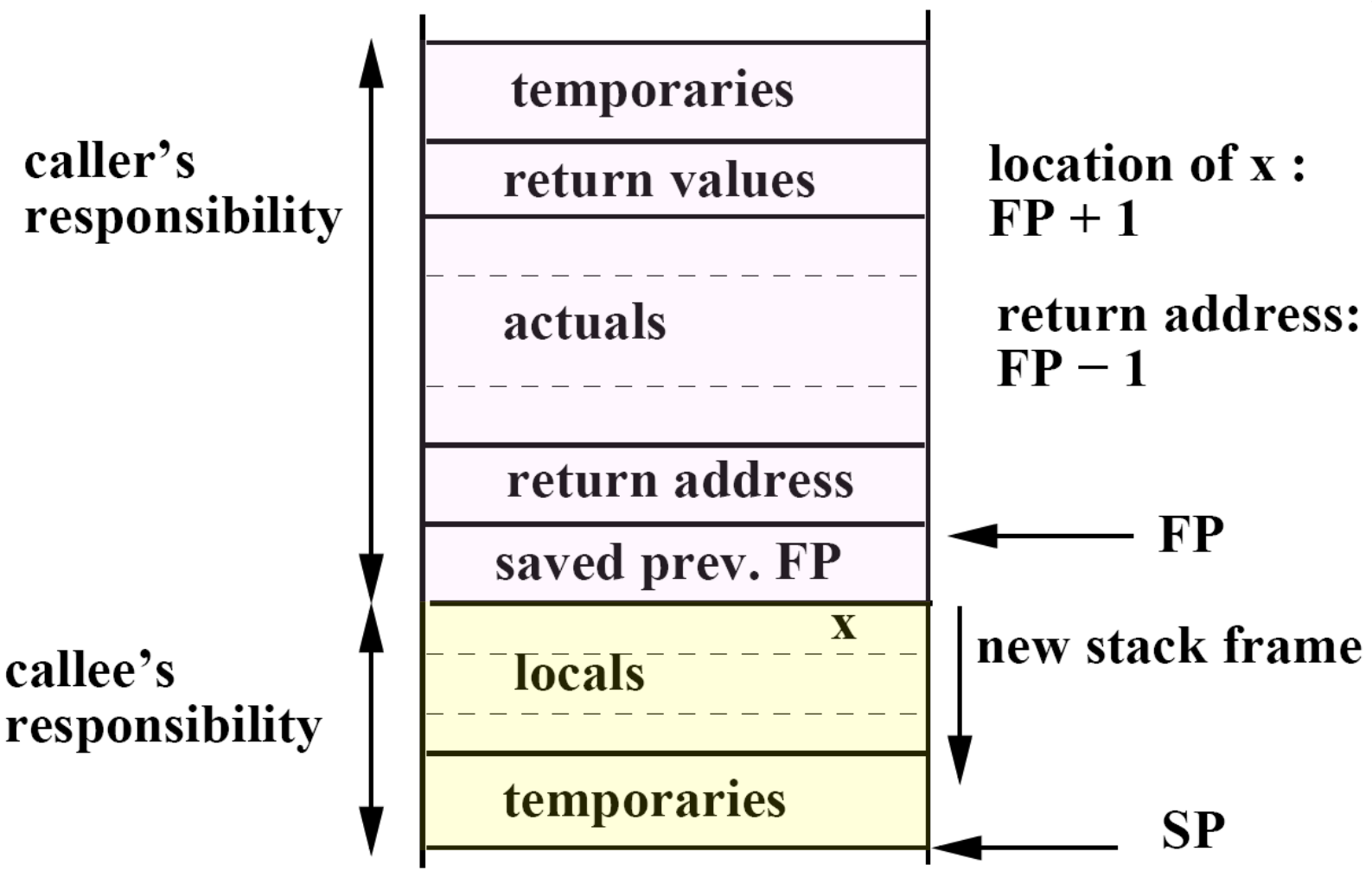
Function

각 함수의 이름을 label로 생성한뒤, 함수 호출시에는 control 명령어를 사용해서 이동

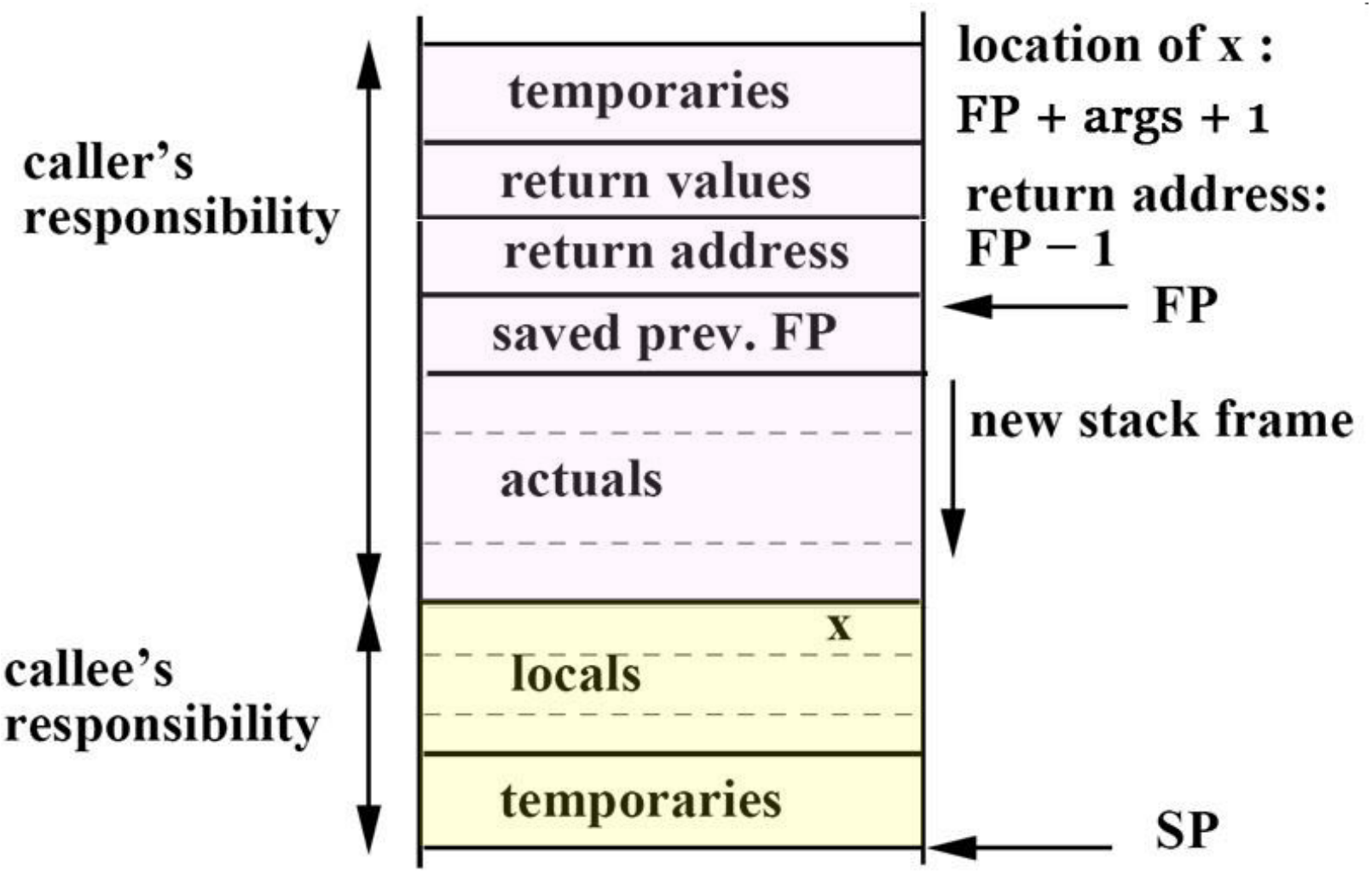
강의 교재의 함수 부분을 참고해서 자신만의 calling convention을 만들기

지역 변수를 위한 스택 공간 할당은 shift_sp를 사용

Example: Calling Convention I



Example: Calling Convention II



Deeper Implementation

심화 구현의 경우 예제 코드가 제공되지 않음

while, for문

- nested for문은 없다고 가정
- nested while문은 고려해야 함
- break, continue도 구현

struct의 연산

- assignment, return, parameter

구현 했을 경우 결과 보고서에 반드시 작성

- 각각의 구현에 대해 자세한 설명이 필요함
- 작성하지 않은 경우 0점 처리

EXAMPLES

Example - HelloWorld

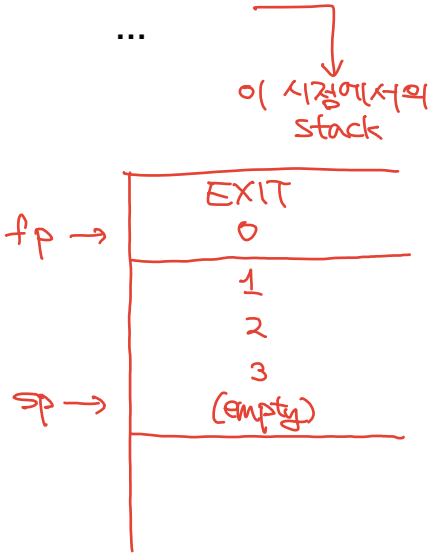
```
            shift_sp 1
            push_const EXIT
            push_reg fp
            push_reg sp
            pop_reg fp
            jump main
EXIT:
            exit
main:
main_start:
Str0. string “hello world\n”
            push_const Str0
            write_string
main_exit:
            push_reg fp
            pop_reg sp
            pop_reg fp
            pop_reg pc
main_end:
Lglob. data 0
```

```
int main() {
    write_string(“hello world\n”);
}
```

```
smb1221@capella:~/compiler/sim$ ./sim t.s
code area size 12
data area size 14
hello world
program exits
```

Example - func2(caller)

```
main:
        shift_sp 4
main_start:
        push_reg fp
        push_const 1
        add
        push_reg sp
        fetch
        push_const 1
        assign
        fetch
        shift_sp -1
        push_reg fp
        push_const 2
        add
        push_reg sp
        fetch
        push_const 2
        assign
        fetch
        shift_sp -1
        push_reg fp
        push_const 3
        add
        push_reg sp
        fetch
        push_const 3
        assign
        fetch
        shift_sp -1
        …
```




```
int test(int a, int b, int c){
    return a;
}

int main(){
    int i;
    int j;
    int k;
    int l;

    i = 1;
    j = 2;
    k = 3;

    l = test(i, j, k);
}
```

Example - func2(caller)

```
        …
        push_reg fp
        push_const 4
        add
        push_reg sp
        fetch
        shift_sp 1
        push_const label_0
        push_reg fp
        push_reg fp
        push_const 1
        add
        fetch
        push_reg fp
        push_const 2
        add
        fetch
        push_reg fp
        push_const 3
        add
        fetch
        push_reg sp
        push_const -3
        add
        pop_reg fp
        jump test
label_0:
        assign
        fetch
        shift_sp -1
Lglob.  data 0
```

```
int test(int a, int b, int c){
    return a;
}

int main(){
    int i;
    int j;
    int k;
    int l;

    i = 1;
    j = 2;
    k = 3;

    l = test(i, j, k);
}
```

Example - func2(callee)

```
test:
test_start:
    push_reg fp
    push_const -1
    add
    push_const -1
    add
    push_reg fp
    push_const 1
    add
    fetch
    assign
    jump test_final
test_final:
    push_reg fp
    pop_reg sp
    pop_reg fp
    pop_reg pc
test_end:
```

return value addr

a 가져옴

← return value 넣기

callee's resp.

→ fp

→ prev fp

→ return label.

```
int test(int a, int b, int c){
    return a;
}

int main(){
    int i;
    int j;
    int k;
    int l;

    i = 1;
    j = 2;
    k = 3;

    l = test(i, j, k);
}
```

Example - struct1

```
main:
        shift_sp 84
main_start:
        push_reg fp
        push_const 1
        add
        push_reg sp
        fetch
        push_const 7
        assign
        fetch
        shift_sp -1
        push_reg fp
        push_const 5
        add
        push_reg fp
        push_const 1
        add
        fetch
        push_const 8
        mul
        add
        push_const 1
        add
        push_reg sp
        fetch
        push_reg fp
        push_const 1
        add
        fetch
        push_const 10
        sub
        assign
        fetch
        shift_sp -1
Lglob.  data 10
```

```
int global_1;
int global_2;

struct _str1{
   int x;
   int y;
   struct _st2{
      int z;
      int w[5];
   } strstr;
} sample_str;

int main(){
   int i;
   int j;
   int k;
   int *l;
   struct _str1 teststr[10];
   i = 7;
   teststr[i].y = i - 10;
}
```

Example - if

```
main:
        shift_sp 3
main_start:
        . . . . . .
label_0:
        push_reg fp
        push_const 1
        add
        fetch
        push_reg fp
        push_const 2
        add
        fetch
        equal
        branch_false
label_1
        push_reg fp
        push_const 3
        add
        push_reg sp
        fetch
        push_const 1
        assign
        fetch
        shift_sp -1
        jump label_2
label_1:
        push_reg fp
        push_const 3
        add
        push_reg sp
        fetch
        push_const 0
        assign
        fetch
        shift_sp -1
label_2:
Lglob.  data 0
```

```
int main(){
    int a;
    int b;

    int x;

    a = 1;
    b = 2;

    if (a == b) {
        x = 1;
    } else {
        x = 0;
    }
}
```

Example

a++ (a는 int형 전역 변수, 오프셋 0 가정)

```
push_const Lglob
fetch
push_const Lglob
push_const Lglob
fetch
push_const 1
add
assign
```

TIPS&SUBMISSION

Grammar

Code Generation시에 고려하지 않아도 되는 사항들

- Syntax Error, Semantic Error 가 발생하는 코드
- NULL 이 사용되는 코드
- 자기 자신을 call하는 함수, 자기 자신을 멤버로 갖는 구조체
- Char pointer string (e.g. char* a = "Hello";)
  - 따라서 write_string의 경우 `write_string("strings\n");` 형태로의 사용만 고려
  - `char* s = "strings\n"; write_string(s);` 와 같은 경우는 고려 X
- 배열과 포인터, 배열과 배열간 operation
  - ex)
```
int *a;
int arr[3];
a = arr;
```
- Variable Length Array

Score

채점 방식

- 문법적(Syntax, Semantic)으로 아무 문제가 없는 소스코드가 입력됨
  - Project #3에서의 에러 체크를 할 필요는 없음
- `./subc (source .c file_input) (assembly .s file_output)` 의 형식으로 코드를 생성하도록 구현할 것 ex) ./subc test.c output.s
- 생성된 코드를 simulator에 입력으로 주고 출력 결과를 비교

배점

- 기본 구현 (75%)
- 심화 구현 (25%)
  - for, while, struct assignment, struct return, struct parameter 각 5%
- 보고서 미제출 및 실행 방법이 올바르지 않은 경우 감점

Submission

제출 기한

- **12월 20일(월) 23시 59분**
- 클레임 일정: 12/21(화) ~ 12/22(수)

제출 방법

- etl.snu.ac.kr을 통해서 제출

제출 파일

- subc.l, subc.y, subc.h, hash.c, hash.h 등 소스파일과 Makefile, readme 파일, 결과 보고서를 압축해서 zip파일로 제출
- 파일명: project4_학번.zip
- readme 파일에는 이름, 학번, 이메일, 실행방법(Makefile을 변경하였을 경우)을 적는다.

Notice

수업 게시판 확인

- 수정 또는 추가되는 사항은 항상 게시판을 통하여 공지
- 제출 마지막날까지 공지된 사항을 반영해서 제출

eTL 질문 게시판 활용

소스코드에 자세히 주석달기

Cheating 금지 (F처리, 모든 코드 철저히 검사)

TA

- 조중하 (301동 819호)
- e-mail: zoonghi@snu.ac.kr