제6장
함수호출과 어셈블리 언어

Youtube 주소
[6-1] http://youtu.be/5-RhIyaZ6eM
[6-2] http://youtu.be/iMO1rufdxho
[6-3] http://youtu.be/n7OsbawpHIM
[6-2(추가)] http://youtu.be/ace-tPdS1_Q
[6-3(추가1)] http://youtu.be/VB83x3cZCm8
[6-3(추가2)] http://youtu.be/LE-IiaOvfg

C 언어의 함수 호출과 반환 값

```
int x, y;

main()
{
    x = abs(x);
    y = abs(y);
    ...
}

int abs(int a)
{
    if (a < 0)
        return -a;
    return a;
}
```

- 인수 전달 방법
- 반환 값 전달 방법
- 되돌아올 주소 저장 방법

```
main:
  ...
  BR nzp, abs
  STORE R0, x
  ...
  BR nzp, abs
  STORE R0, y
  ...
abs:
  ...
  LOAD R0, ...
  RET
x: .BLOCK 1
y: .BLOCK 1
```

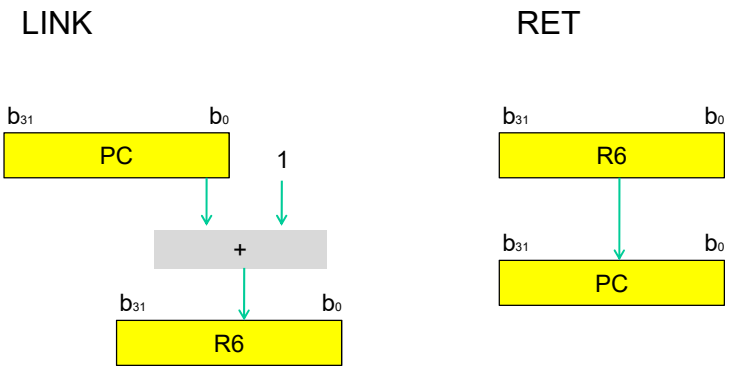
함수 호출을 위한 명령어

이름	어셈블리언어 표현	의미
Link	LINK	되돌아올 주소를 R6에 저장 (R6 ← PC+ 1)
Branch	BR nzp, addr	무조건 addr로 다음 실행위치를 이동 (PC ← addr)
Return	RET	R6에 저장된 주소로 다음 실행위치를 이동 (PC ← R6)

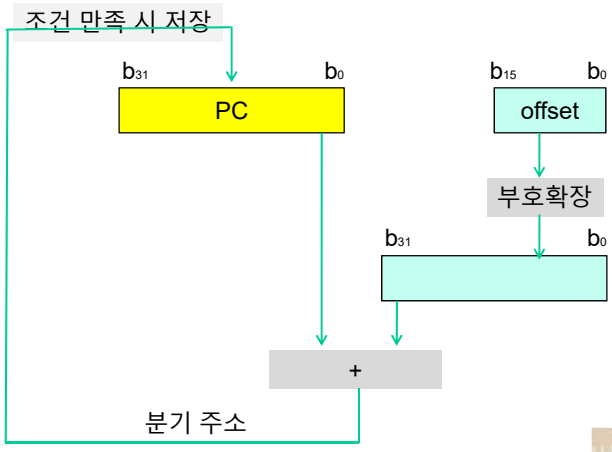
함수 호출과 반환

```
   .ORIGIN 0x2000
main:       ; main 함수
   ...
   LOAD R0, x          ; 변수 x의 값 읽기
   LINK                ; 다음의 BR 바로 다음 주소를 R6에 저장
   BR nzp, abs         ; 함수 abs의 주소로 실행위치 이동
   STORE R0, x         ; 계산된 절대 값을 변수 x에 쓰기
   LOAD R0, y          ; 변수 y의 값 읽기
   LINK                ; 다음의 BR 바로 다음 주소를 R6에 저장
   BR nzp, abs         ; 함수 abs의 주소로 실행위치 이동
   STORE R0, y         ; 계산된 절대 값을 변수 y에 쓰기
   ...
   RET
abs:        ; abs 함수
   CMP R0, 0           ; 전달받은 R0의 값을 0과 비교
   BR zp, out
   COPY R1, 0
   SUB R0, R1, R0      ; R0 ← 0 - R0 (반대 부호로 변환)
out: RET               ; 원래의 실행위치로 복귀
```

LINK / RET 명령어의 실행



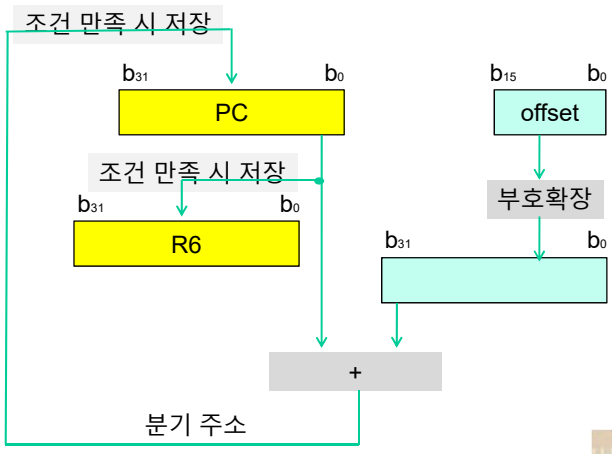
BR 명령어의 실행



(참고) BRL (LINK+BR) 명령어가 있다면?

```
   .ORIGIN 0x2000
main:       ; main 함수
  ...
  LOAD R0, x       ; 변수 x의 값 읽기
  BRL nzp, abs     ; 다음주소를 R6에 저장후 abs의 주소로 이동
  STORE R0, x      ; 계산된 절대 값을 변수 x에 쓰기
  LOAD R0, y       ; 변수 y의 값 읽기
  BRL nzp, abs     ; 다음주소를 R6에 저장후 abs의 주소로 이동
STORE R0, y        ; 계산된 절대 값을 변수 y에 쓰기
  ...
  RET
abs:        ; abs 함수
  CMP R0, 0        ; 전달받은 R0의 값을 0과 비교
  BR zp, out
  COPY R1, 0
  SUB R0, R1, R0
out: RET           ; 원래의 실행위치로 복귀
```


(참고) BRL (LINK+BR) 실행 회로 추가?



함수의 인수와 반환 값 전달

- 인수 전달
  - ✓(방안1) 레지스터 활용: R0, R1, … 의 순서로 사용
  - ✓(방안2) 스택의 활용: 6.6절 참조
- 반환 값의 전달
  - ✓레지스터 활용: R0

2개의 인수를 갖는 함수호출 예

```
int val, x;
main()
{
   ...
   val = min(x, 3);
   ...
}
int min(int a, int b)
{
   if (a < b)
      return a;
   return b;
}
```

C 언어의 함수호출 처리

```
main:
     ...
     LOAD R0, x        ; 첫 번째 인수
     COPY R1, 3        ; 두 번째 인수
     LINK
     BR nzp, min       ; 함수 min 호출
     STORE R0, val     ; 반환 값을 변수 val에 저장
     ...
min:                ; 함수 min
     CMP R0, R1        ; 두 인수의 비교
     BR pz, _L0
     RET               ; R0 값을 그대로 반환
_L0:
     COPY R0, R1       ; R0 에 두 번째 인수를 복사하여
     RET               ; R0 값을 반환
val: .BLOCK 1
x:   .BLOCK 1
```


정수의 곱하기 함수 MUL

```
int MUL(int a, int b)
// assume b > 0
{
   int result;

   result = 0;
   while (b > 0)
   {
      result = result + a;
      b = b - 1;
   }
   return result;
}
```

```
while (--b >= 0)
   result += a;
```

정수의 곱하기 함수 MUL

- MUL(a, b);
- R0 ← R0 x R1 (R1은 양수라고 가정)

```
MUL:    ; R0의 값과 R1의 값을 곱해서 R0에 저장한 후 반환
    COPY R2, 0      ; R2를 곱의 결과 저장용으로 사용
_L1: SUB R1, R1, 1
    BR n, _L2
    ADD R2, R2, R0
    BR nzp, _L1
_L2: COPY R0, R2    ; R0에 반환할 값 기록
    RET
```


정수의 나누기 함수 DIV

- DIV(a, b);
- R0 ← R0 / R1 (R0, R1 모두 양수라고 가정)

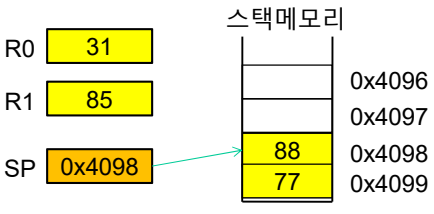
```
DIV:    ; R0의 값에서 R1을 나눈 몫을 R0에 저장한 후 반환
    COPY R2, 0      ; R2: 횟수 기록용
_L4: ADD R2, R2, 1
    SUB R0, R0, R1
    BR zp, _L4
    SUB R0, R2, 1   ; 반환값(R0)은 마지막 증가분을 취소한 값
    RET
```

함수 MUL과 DIV를 활용한 수식 연산

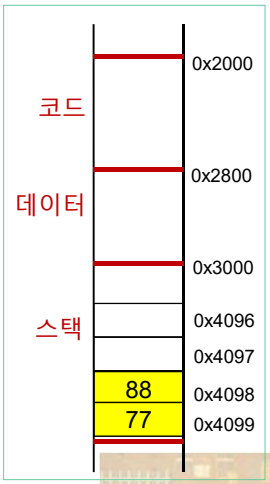
```
; C 언어로 "z = x * (y+2) / 10;"을 위한 프로그램
    LOAD R0, x
    LOAD R1, y
    ADD R1, R1, 2
    LINK
    BR nzp, MUL       ; 곱한 결과는 R0에 저장
    COPY R1, 10
    LINK
    BR nzp, DIV       ; 나눈 결과는 R0에 저장
    STORE R0, z
    ...
MUL:
    ...
DIV:
    ...
```


스택 (Stack) 의 사용

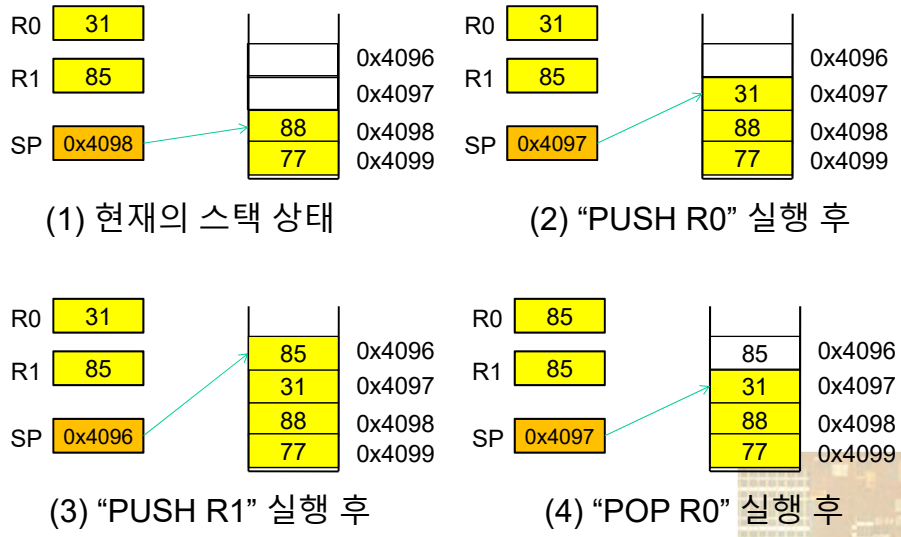
- LIFO (Last In First Out)
- PUSH 와 POP 동작으로 구성
  - ✓PUSH:
  - ✓POP:
- 스택의 상단을 가리키는 레지스터 필요
  - ✓SP (Stack Pointer) → TOY에서는 R5



스택의 현재 상태



스택 동작 시나리오 예



TOY의 PUSH 와 POP 처리

- TOY는 PUSH와 POP을 위한 별도 명령어가 없음
  - SUB, STR, LDR, ADD 명령어 등을 조합하여 처리
- 스택포인터는 R5를 사용

```
; “PUSH Rs” 동작
    SUB R5, R5, 1      ; 스택포인터 값을 1만큼 감소
    STR Rs, R5, 0      ; 스택 상단에 Rs의 값 저장
; “POP Rd” 동작
    LDR Rd, R5, 0      ; 스택 상단에서 읽어서 Rd에 기록
    ADD R5, R5, 1      ; 스택포인터 값을 1만큼 증가
```

스택을 이용한 링크 레지스터의 저장과 복구

- 함수에서 다른 함수를 호출하면
  - 링크 레지스터 값이 변경됨

  → 링크 레지스터 값을 함수호출 전에 저장

```
f1:
    SUB R5, R5, 1     ; "PUSH R6"
    STR R6, R5, 0
    ...
    LINK
    BR nzp, f2
    ...
    LDR R6, R5, 0     ; "POP R6"
    ADD R5, R5, 1
    RET

f2:
    ...
    RET
```


스택을 이용한 인수 전달

- 함수호출 시 전달할 인수를 스택에 PUSH

  ✓역순으로 PUSH
- 함수에서는 스택에 저장된 인수를 사용
- 프로그램 6.4: 레지스터를 이용한 인수 전달
- 프로그램 6.10: 스택을 이용한 인수 전달

스택을 이용한 인수의 전달 (프로그램 6.10)

```
main:
    ...
    COPY R0, 3          ; 두 번째 인수
    SUB R5, R5, 1       ; 'PUSH R0' (두 번째 인수 푸시)
    STR R0, R5, 0
    LOAD R0, x          ; 첫 번째 인수
    SUB R5, R5, 1       ; 'PUSH R0' (첫 번째 인수 푸시)
    STR R0, R5, 0
    LINK
    BR nzp, min         ; 함수 min 호출
    ADD R5, R5, 2       ; 2번의 푸시에서 감소한 만큼 복구
    STORE R0, val       ; 반환 값을 변수 val에 기록
    ...
min:      ; 함수 min
    LDR R0, R5, 0       ; 첫 번째 인수 가져오기
    LDR R1, R5, 1       ; 두 번째 인수 가져오기
    CMP R0, R1
    BR pz, _L0
    RET                 ; R0 값을 그대로 반환
_L0:
    COPY R0, R1         ; R0 에 두 번째 인수를 복사하여
    RET                 ; R0 값을 반환
val: .BLOCK 1
x:   .BLOCK 1
```

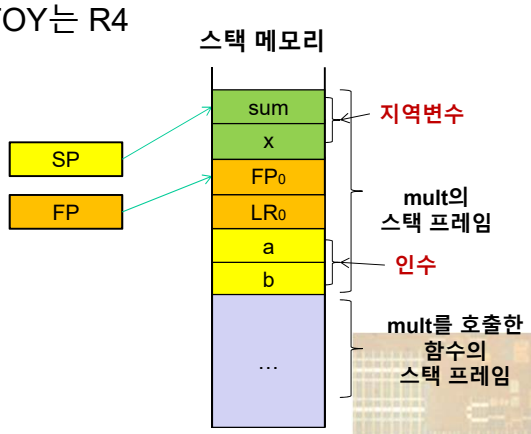

지역 변수를 갖는 함수

```
int mult(int a, int b)
{
    int x, sum;

    sum = 0;
    x = b;
    while (x > 0) {
        sum = sum + a;
        x = x - 1;
    }
    return sum;
}
```

함수별 스택 프레임

- 함수별 스택 프레임: 함수의 실행 중에 인수 및 지역변수들을 포함하는 스택 메모리 영역
- FP (Frame Pointer): 스택 프레임의 기준이 되는 주소를 저장하는 레지스터 → TOY는 R4
- 함수 시작부분에서
  - PUSH LR
  - PUSH FP
  - FP ← SP

```
int mult(int a, int b)
{   int x, sum;
    ...
}
```



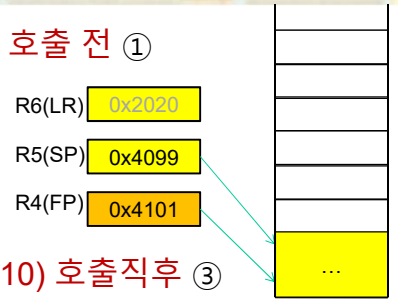
함수에서 스택 프레임의 조정과 복구

```
   ; int mult(int a, int b)
   ; {
mult:
③  SUB R5, R5, 2       ; SP 값 조정
    STR R6, R5, 1       ; 현재의 LR 값 PUSH
    STR R4, R5, 0       ; 현재의 FP 값 PUSH
    COPY R4, R5         ; SP 값을 FP에 복사
   ; int x, sum;
④  SUB R5, R5, 2 ; x와 sum을 위한 스택 공간
⑤
    ...

   ; }
⑥  COPY R5, R4         ; SP 값 복구
    LDR R6, R5, 1       ; LR 값 복구
    LDR R4, R5, 0       ; FP 값 복구
    ADD R5, R5, 2       ; SP 값 조정

⑦  RET
```

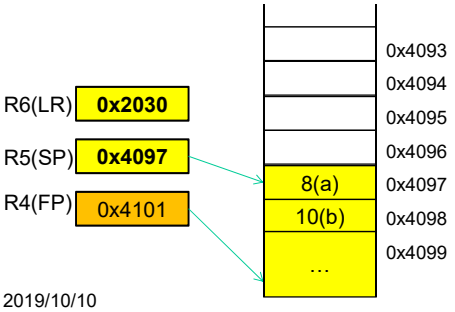
```
    ...
; mult 함수의 호출 예
; value = mult(8,10)
① COPY R0, 10
  SUB R5, R5, 1
  STR R0, R5, 0
  COPY R0, 8
  SUB R5, R5, 1
  STR R0, R5, 0
② LINK
  BR nzp, mult
⑧ ADD R5, R5, 2
⑨ STORE R0, value
  ...
```

Mult 함수에서 FP와 SP의 조정

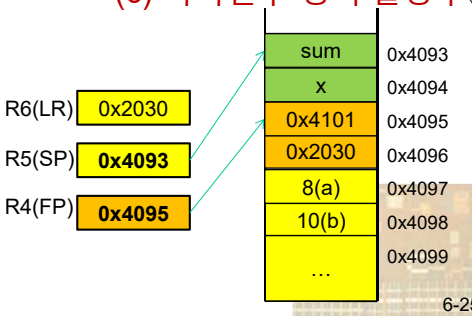
(a) mult 호출 전 ①



(b) mult(8,10) 호출직후 ③

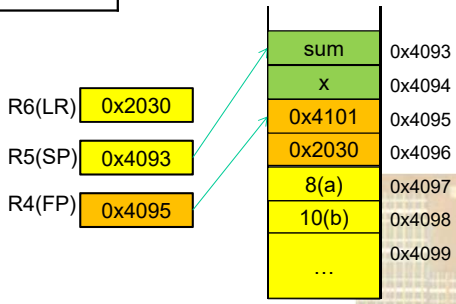


(c) 지역변수 영역 할당후⑤

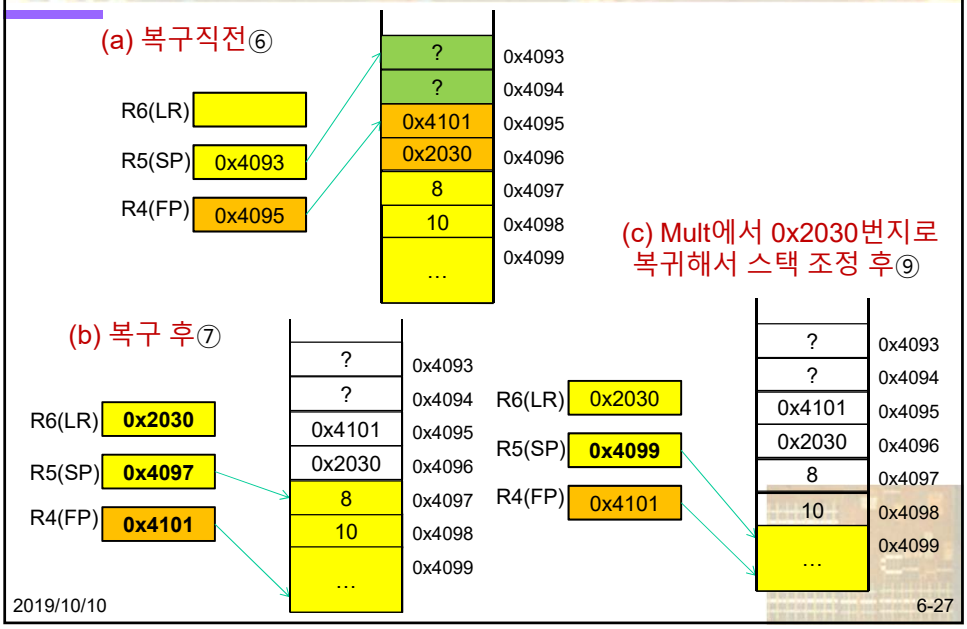


함수 mult에서 인수 및 지역변수 접근

인수 및 지역변수	메모리 상의 주소	R0에 값 읽어 오기
a	R4 + 2	LDR R0, R4, 2
b	R4 + 3	LDR R0, R4, 3
x	R4 - 1	LDR R0, R4, -1
sum	R4 - 2	LDR R0, R4, -2



Mult 함수에서 FP와 SP의 복구



Mult 함수의 컴파일 결과

- 프로그램 6.14

```
; int mult(int a, int b)
; {
mult:
            SUB R5, R5, 2
            STR R6, R5, 1       ; 현재의 LR 값 PUSH
            STR R4, R5, 0       ; 현재의 FP 값 PUSH
            COPY R4, R5                 ; SP 값을 FP에 복사
;           int x, sum;
            SUB R5, R5, 2               ; 지역변수 x와 sum을 위한 스택 공간
;           sum = 0; x = b;
            COPY R0, 0
            STR R0, R4, -2              ; 지역변수 sum에 쓰기
            LDR R0, R4, 3               ; 지역변수 b 읽기
            STR R0, R4, -1              ; 지역변수 x에 쓰기
;           while (x > 0) {
_L3:
            LDR R0, R4, -1              ; 지역변수 x 읽기
            COPY R1, 0
            CMP R0, R1
            BR nz, _L4
;                       sum = sum + a;
            LDR R0, R4, -2              ; 지역변수 sum 읽기
            LDR R1, R4, 2 ; 인수 a 읽기
            ADD R0, R0, R1
            STR R0, R4, -2              ; 지역변수 sum에 쓰기
;                        x = x - 1;
            LDR R0, R4, -1              ; 지역변수 x 읽기
            SUB R0, R0, 1
            STR R0, R4, -1              ; 지역변수 x에 쓰기
;           }
            BR nzp, _L3
_L4:
;           return sum;
            LDR R0, R4, -2              ; 지역변수 sum 읽기 (함수의 반환 값)
; }
_L2:        COPY R5, R4                 ; SP 값 복구
            LDR R4, R5, 0 ; FP 값 복구
            LDR R6, R5, 1 ; LR 값 복구
            ADD R5, R5, 2
            RET
```

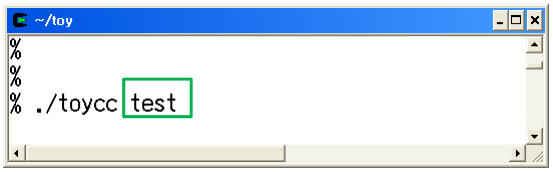
TOY C 컴파일러

- 함수의 인수 전달: 스택을 통해 전달하며, 역순으로 PUSH
- 반환값 전달: R0 사용
- 스택 포인터: R5 사용
- 프레임 포인터: R4 사용
- 지역변수는 스택 상단에 할당
- 함수 시작부분에 링크 레지스터를 PUSH, 함수 끝부분에서 POP

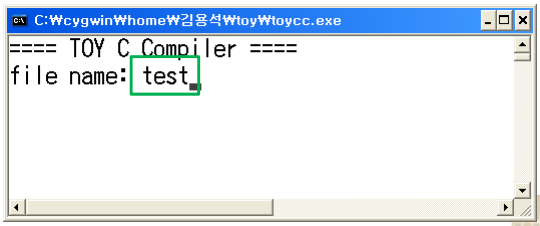
TOY C 컴파일러의 실행

test.c 의 컴파일

명령창에서 파일이름 입력



실행화면에서 파일이름 입력



toycc로 컴파일한 예

test.c (프로그램 5.1)

```
int x, y, z;
main()
{
  x = 2; y = 3;
  z = x + y - 4;
}
```

test.s (프로그램 5.2)

```
    .ORIGIN 0x2000
_CODE:
  ; program startup code
    LEA R5, _END
    ADD R5, R5, 1024
    LINK
    BR nzp, main
    SWI 255
  ; main()
  ; {
main:
    SUB R5, R5, 2
    STR R6, R5, 1
    STR R4, R5, 0
    COPY R4, R5
  ; x = 2; y = 3;
    COPY R0, 2
    STORE R0, x
    COPY R0, 3
    STORE R0, y
  ; z = x + y - 4;
    LOAD R0, x
    LOAD R1, y
    ADD R0, R0, R1
    SUB R0, R0, 4
    STORE R0, z
  ; }
_L1:COPY R5, R4
    LDR R4, R5, 0
    LDR R6, R5, 1
    ADD R5, R5, 2
    RET
_DATA:
  ; int x, y, z;
x: .BLOCK 1
y: .BLOCK 1
z: .BLOCK 1

_END:
```

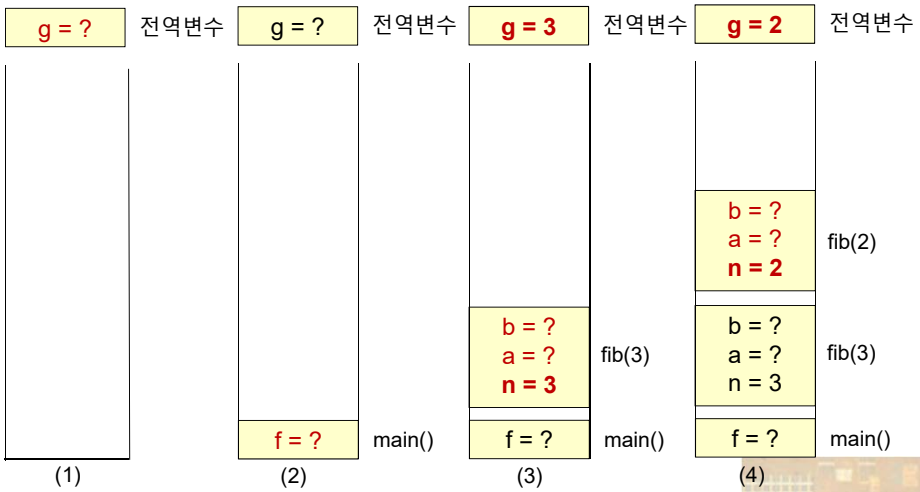

toycc의 컴파일 결과 예

- test.c (프로그램 5.1) / test.s (프로그램 5.2)
- c369.c (프로그램 6.15) / c369.s (프로그램 6.16)
- fib.c (프로그램 6.17) / fib.s (프로그램 6.18)

재귀함수에서 인수와 지역변수 처리 예

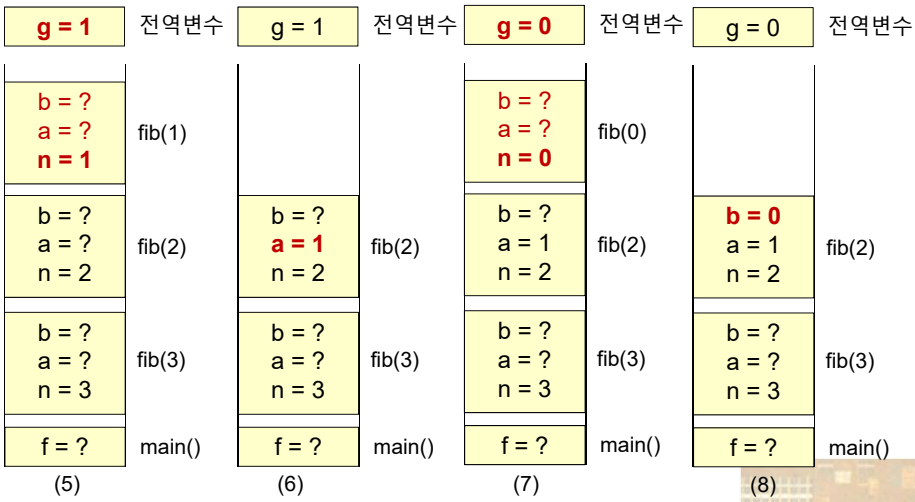
```
int g;              // g: global variable
int main()
{
   int f;           // f: local variable
   f = fib(3);
   g = f;
}

int fib(int n)      // n: argument
{
   int a, b;        // a,b: local variable
   g = n;
   if (n <= 1)
       return n;
    a = fib(n-1);
    b = fib(n-2);
   return a+b;
}
```


스택 프레임의 변화 (1/3)



(참고) 지역변수는 함수 별로 있는 것이 아니고, ***함수호출****마다 별도로 생성*

스택 프레임의 변화 (2/3)



스택 프레임의 변화 (3/3)