1. goto

goto문을 만나면 프로그래머가 지정한 장소로 점프해서 이동한다. if문은 어셈블리로 변환하면 mov, cmp, jmp 3단계를 거치지만 goto문은 jmp 한단계만을 거친다. 그러므로 성능이 더 좋다.

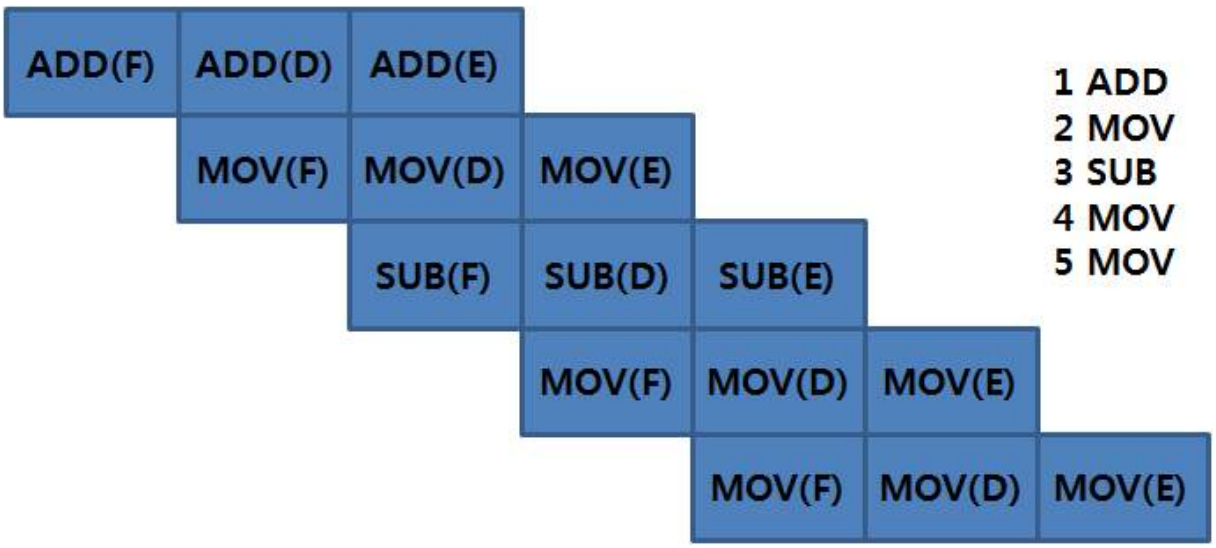
또한, 프로그램이 다중 for문 구조인 경우에 가장 안쪽 for문에서 특정조건을 만족하면 모든 for문을 벗어나고자 할 때, break를 사용하는 경우와 goto를 사용하는 경우 2가지가 있다. 일반적으로 프로그래머들에게는 goto 보다는 break가 더 익숙하다. 하지만 break을 사용하게 되면 가장 안쪽 for문에서 모든 for문을 벗어나고자 할 때, for문의 개수만큼 break가 필요하다. 이것은 프로그램이 외관상으로 복잡해지고 가독성도 떨어뜨린다. 반대로 goto를 사용하면 가장안쪽 for문에서 한번만 사용하면 된다. 그러므로 break보다 가독성도 높고 성능이 좋다. 또한 goto를 만났을 때 이동할 곳을 프로그래머가 마음대로 지정할 수 있는 장점까지 있다. 비록 많은 프로그래머들에게 익숙하지 않더라도 적극적으로 사용할 필요가 있다.

goto나 break, continue 등의 명령어들은 함수가 아니다. 어셈블리어(jmp) 명령어이다.

2. 파이프라인

파이프라인은 CPU가 명령의 fetch, decode, execute 단계를 병렬적으로 수행하는 방식을 의미한다. 예를 들어, cpu가 3단계 파이프라인의 간단한 아키텍처이고, 오른쪽의 5개의 명령어를 실행한다고 했을 때, 파이프라인의 동작은 아래 그림과 같다. (F=fetch, D=decode, E=execute) fetch, decode, execute의 3단계 동작을 3개의 명령어를 한꺼번에 실행하는 구조이다. 만약 cpu가 이러한 파이프라인 구조를 가지지 않고, 하나의 명령어의 수행을 끝내고(fetch,decode,execute) 다음 명령어를 수행한다면 굉장히 시간이 오래걸릴 것이다.

만약 파이프라인의 단계가 늘어난다면 명령어 수행 단계가 늘어나는 것이다. fetch, decode, execute외에 다른 단계가 더 추가된다. 그러면 한번에 수행할 수 있는 명령어 갯수도 그만큼 늘어나게 되므로 더 효율적이게 된다.



3. for문

for문의 기본 구조는 **for(초기식;조건식;증감식) 명령;** 이다. 초기식을 실행하고 루프가 돌기 시작해서 한번 돌 때마다 증감식을 실행한다. 조건식이 참인 동안 계속 루프를 돌고 조건식이 거짓이 되면 루프를 탈출한다.

for문과 while문은 상당히 비슷하다고 할 수 있다. 사실 for문이 없어도 while문으로 대체할 수 있다. 반대로 while문이 없어도 for문으로 대체할 수 있다. 이러한 사실에도 불구하고 for문, while문 모두 사용하는 이유는 프로그램 논리 구조가 각각에 더 적합한 상황들이 있기 때문이다. 예를 들어, 정해진 횟수만큼 반복해야 하는 경우에는 주로 for문을 사용한다. 그렇지만 계속 반복수행하다가 어떤 특정한 조건이 되었을 때 멈추는 경우는 while문안에 if과 break를 함께 사용하거나 또는 goto를 사용한다.

- for문 예시

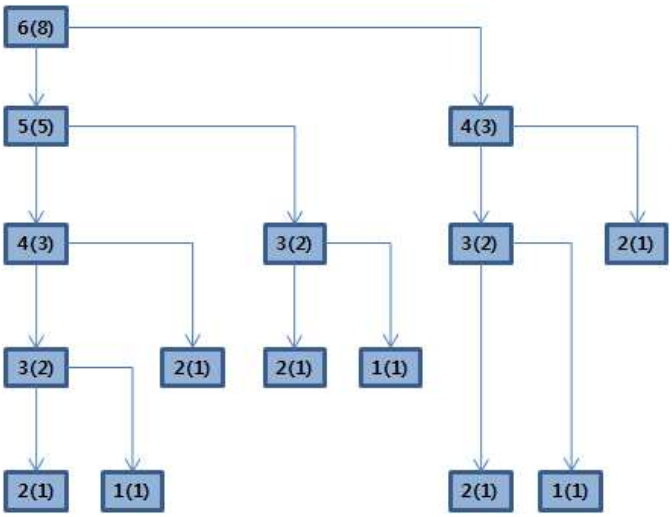
```
for(i=0;i<10;i++)
{
        printf("안녕하세요!\n");
}
```

i가 0부터 루프를 한번 돌때마다 1씩 증가해서 10보다 작지않게 되면 for문을 탈출한다. 그러므로 루프는 총 10번 돌게되고 printf("안녕하세요!\n"); 는 총 10번 실행된다.

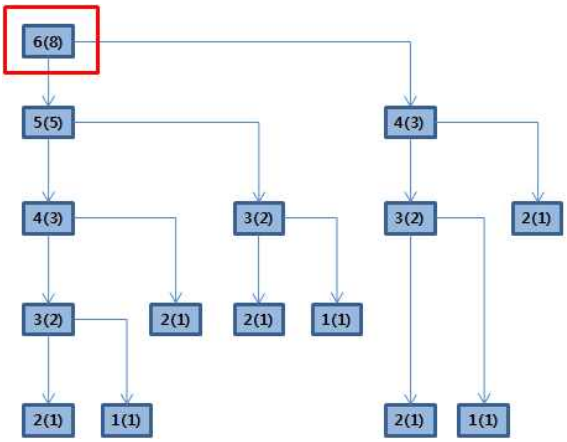
초기식과 조건식, 증감식은 상황에 따라 얼마든지 변형이 가능하다. 또한, for문 안에 for문을 1개이상 넣어서 다중 for문을 만들 수도 있다.

* fib 함수 동작 분석

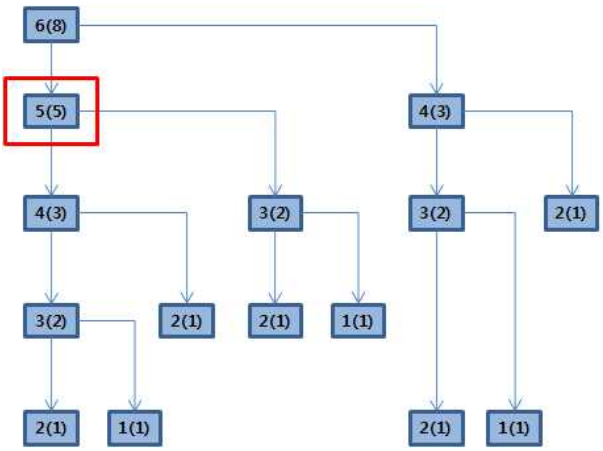
- fib 재귀함수의 전체 동작은 아래 그림과 같다. 사각형은 num이 해당 숫자일 때의 fib함수에 들어간 경우이고, 괄호 안의 숫자는 해당 함수의 return 값이다.



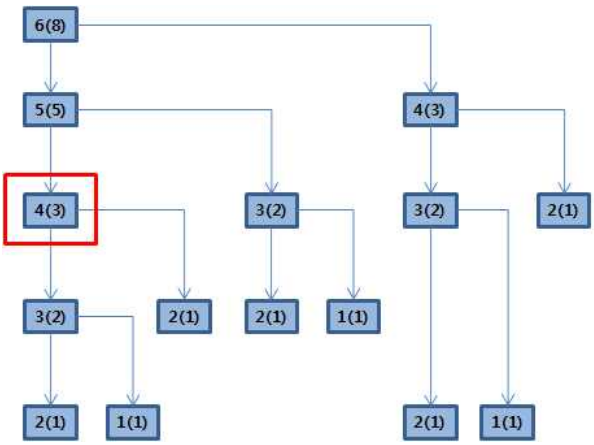
- 디버깅 결과



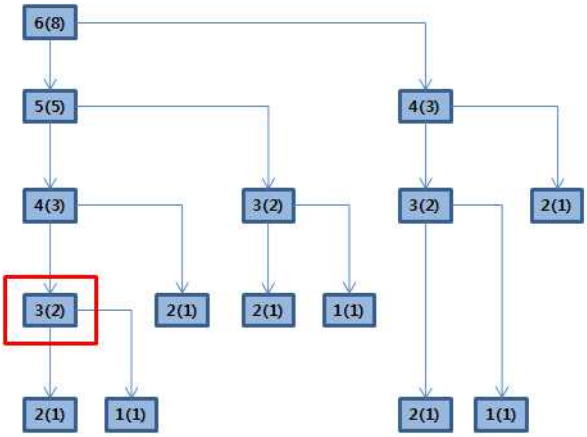

```
#0  fib (num=6) at test.c:22
#1  0x000000000040063c in main () at test.c:11
```



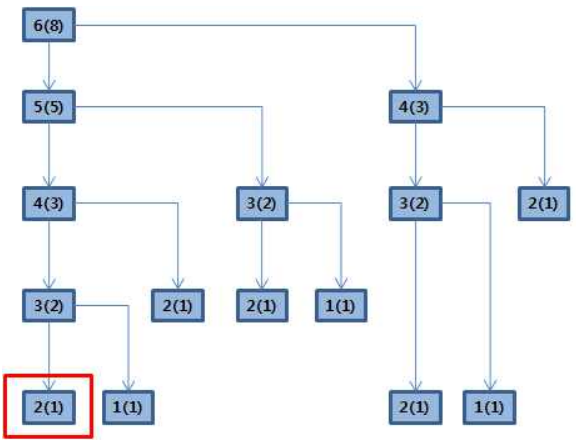

```
#0  fib (num=5) at test.c:19
#1  0x000000000040069d in fib (num=6) at test.c:22
#2  0x000000000040063c in main () at test.c:11
```



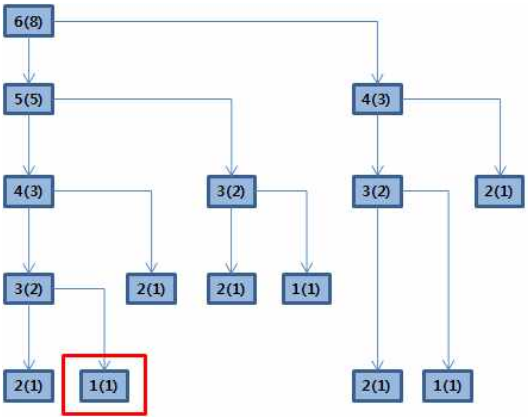

```
#0  fib (num=4) at test.c:19
#1  0x000000000040069d in fib (num=5) at test.c:22
#2  0x000000000040069d in fib (num=6) at test.c:22
#3  0x000000000040063c in main () at test.c:11
```



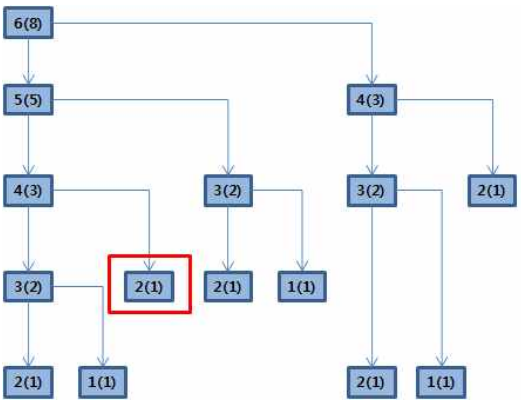

```
#0  fib (num=3) at test.c:19
#1  0x000000000040069d in fib (num=4) at test.c:22
#2  0x000000000040069d in fib (num=5) at test.c:22
#3  0x000000000040069d in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```



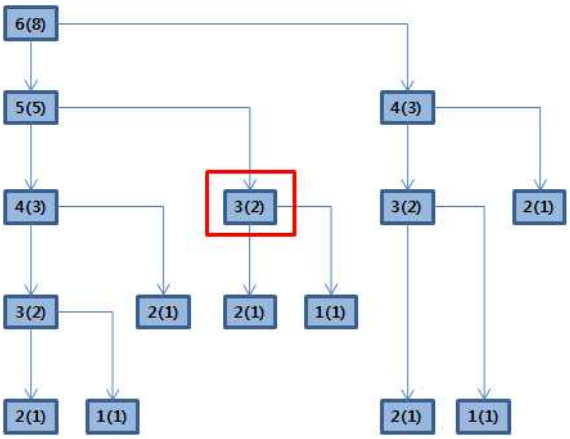

```
#0  fib (num=2) at test.c:19
#1  0x000000000040069d in fib (num=3) at test.c:22
#2  0x000000000040069d in fib (num=4) at test.c:22
#3  0x000000000040069d in fib (num=5) at test.c:22
#4  0x000000000040069d in fib (num=6) at test.c:22
#5  0x000000000040063c in main () at test.c:11
```



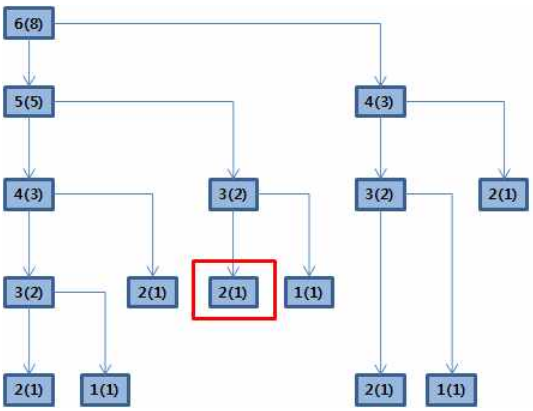

```
#0  fib (num=1) at test.c:19
#1  0x00000000004006ac in fib (num=3) at test.c:22
#2  0x000000000040069d in fib (num=4) at test.c:22
#3  0x000000000040069d in fib (num=5) at test.c:22
#4  0x000000000040069d in fib (num=6) at test.c:22
#5  0x000000000040063c in main () at test.c:11
```



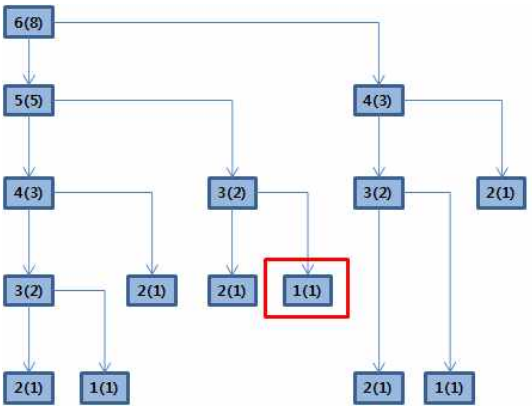

```
#0  fib (num=2) at test.c:20
#1  0x00000000004006ac in fib (num=4) at test.c:22
#2  0x000000000040069d in fib (num=5) at test.c:22
#3  0x000000000040069d in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```



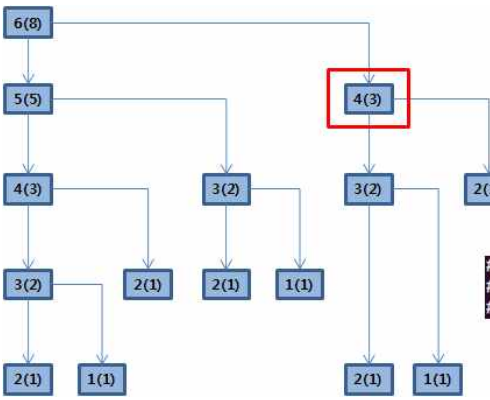

```
#0  fib (num=3) at test.c:19
#1  0x00000000004006ac in fib (num=5) at test.c:22
#2  0x000000000040069d in fib (num=6) at test.c:22
#3  0x000000000040063c in main () at test.c:11
```



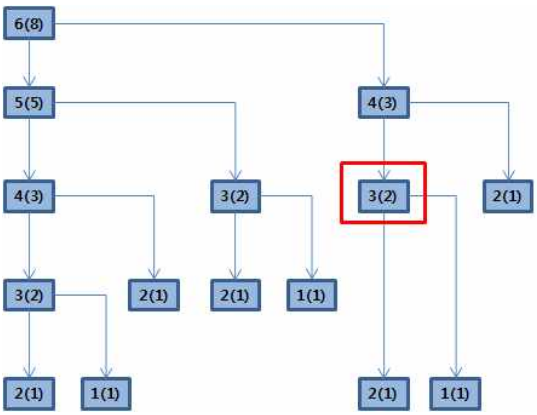

```
#0  fib (num=2) at test.c:19
#1  0x000000000040069d in fib (num=3) at test.c:22
#2  0x00000000004006ac in fib (num=5) at test.c:22
#3  0x000000000040069d in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```



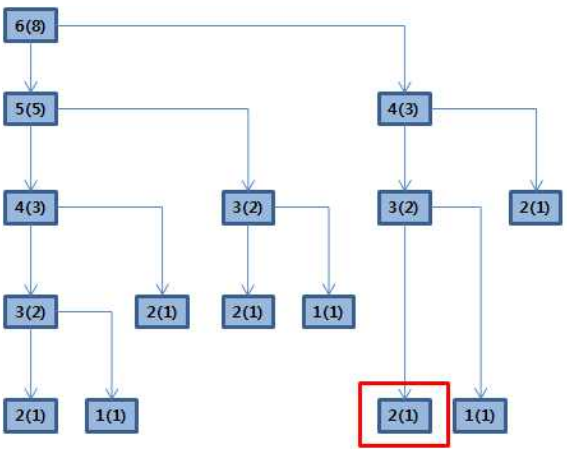

```
#0  fib (num=1) at test.c:19
#1  0x00000000004006ac in fib (num=3) at test.c:22
#2  0x00000000004006ac in fib (num=5) at test.c:22
#3  0x000000000040069d in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```



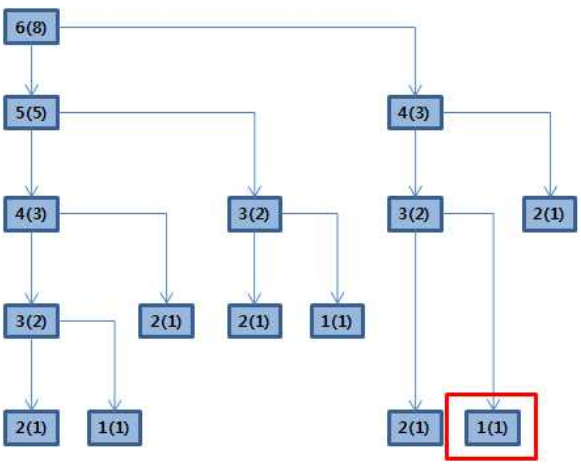

```
#0  fib (num=4) at test.c:19
#1  0x00000000004006ac in fib (num=6) at test.c:22
#2  0x000000000040063c in main () at test.c:11
```



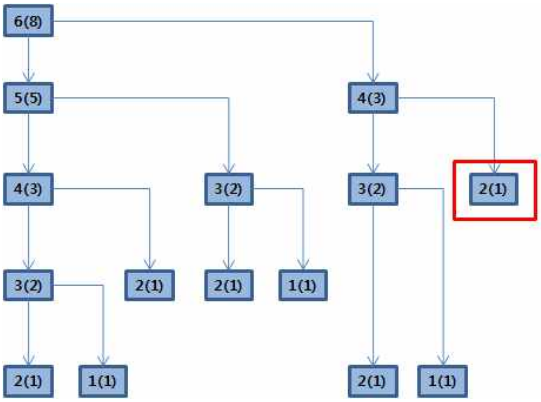

```
#0  fib (num=3) at test.c:19
#1  0x000000000040069d in fib (num=4) at test.c:22
#2  0x00000000004006ac in fib (num=6) at test.c:22
#3  0x000000000040063c in main () at test.c:11
```




```
#0  fib (num=2) at test.c:19
#1  0x000000000040069d in fib (num=3) at test.c:22
#2  0x000000000040069d in fib (num=4) at test.c:22
#3  0x00000000004006ac in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```




```
#0  fib (num=1) at test.c:19
#1  0x00000000004006ac in fib (num=3) at test.c:22
#2  0x000000000040069d in fib (num=4) at test.c:22
#3  0x00000000004006ac in fib (num=6) at test.c:22
#4  0x000000000040063c in main () at test.c:11
```




```
#0  fib (num=2) at test.c:19
#1  0x00000000004006ac in fib (num=4) at test.c:22
#2  0x00000000004006ac in fib (num=6) at test.c:22
#3  0x000000000040063c in main () at test.c:11
```

- 마지막 결과

```
12              printf("%d번째 항의 수는 = %d\n", final_val, result);
(gdb) bt
#0  main () at test.c:12
(gdb) n
6번째 항의 수는 = 8
14              return 0;
```