# README

## 1. 구성

- **n의 값을 입력 받는 문구 출력**

가장 먼저 n의 값을 입력 받는 출력문을 구상하였다. $a0에 "Enter the value of n: " 문자열을 저장한 후 $v0에는 4를 넣어 출력하였다.

- **n의 입력**

$v0에 5를 넣어 레지스터 $s0에 정수를 입력 받도록 프로그램을 구성하였다. 우리가 얻고자 하는 n번째 피보나치 수는 기본 행렬 [1, 1, 1, 0]을 (n-1)제곱한 행렬의 1행 1열의 수이기 때문에 미리 입력 받은 n을 1감소시킨다.

n에 0이 입력된 경우 finish 함수로 jump하여 0을 출력 후 프로그램을 끝내도록 설계하였다.

- **fibonacci 함수**

우선 return address와 나머지 값을 스택에 저장한다. 그리고 행렬 s2에 기존의 행렬 s1 값을 복사 붙여 넣는다. 사실 코드의 흐름상 행렬 s2에 초기 s1의 값인[1, 1, 1, 0]을 대입하는 상황과 동일하다.

$s0에 저장된 변형된 n의 값이 1이하이면 저장된 return address로 jump하고 1보다 크다면 matrixpower라는 함수로 jump한다.

- **matrixpower 함수**

fibonacci 함수에서 matrixpower로 넘어오면 $s0에 저장된 값을 2로 나눈다. 만약 처음으로 fibonacci 함수로 넘어온다면 $s0에는 (n-1)/2의 값이 새롭게 저장되고 $s3에는 나머지가 저장된다.

이후 다시 fibonacci 함수로 돌아가서 2로 나눈 나머지와 (jal fibonacci)다음 명령문의 return address를 스택에 저장하고 $s0의 값이 2보다 작지 않은지 확인한다. 결국은 주어진 (n-1)을 몫이 1이 될 때까지 계속해서 2로 나누고 그때의 나머지와 ra값을 스택에 저장하는 형태이다.

반복해서 2로 나누어진 $s0 값이 1이 되었을 때 fibonacci 함수에서 'jr $ra'라는 명령어를 만난다. 이때 $ra에 저장되어 있는 주소는 matrixpower 함수에서 'jal fibonacci' 다음 명령어인 'jal multiply_power'이다. jal Fibonacci라는 명령어를 통해 $ra에 자동으로 다음 명령어인 'jal

multiply_power'의 주소가 저장되었기 때문이다.

이후 설명할 multiply_power라는 함수로 jump하고 일련의 과정을 거친 다음 'jal multiply_power' 다음 명령어인 'lw $s3, ~' 주소로 return하면 스택에 저장되어 있던 나머지 값을 로드하게 된다. 한 번의 재귀를 거쳤기 때문에 스택에 저장되어 있던 제일 최신의 나누기 과정에서 저장된 나머지와 ra값을 지우고 그 다음 나누기 과정의 return address를 미리 받아 놓는다.

아까 받아 놓은 $s3의 값(2로 나눈 나머지)가 1인 경우 재귀하면서 행렬의 거듭제곱을 할 때 [1, 1, 1, 0]을 한 번 더 곱해야 하는 문제가 발생한다. 따라서 $s3에 저장된 나머지가 1인 경우 multiply_basic 함수로 이동해 기본행렬 [1, 1, 1, 0]을 곱하는 과정을 거치도록 설계하였다.

그 다음의 명령문인 'jr $ra'에서 $ra의 값은 아까 스택에서 받아 놓은 return address이다. 재귀하고 있는 과정에서는 스택에 저장되어 있던 return address는 함수 matrixpower의 'jal multiply_power' 명령문의 주소일 것이다(n-1을 반복적으로 2로 나누는 과정에서 스택에 저장된 return address이다).

만약 재귀가 모두 끝난 상황이라면, 즉, 스택에 처음으로 저장되어(스택 가장 상부에 저장되어 있는) 있던 $ra값은 main 함수의 'jal fibonacci' 다음의 명령어인 'lw $t0, 0($s1)'의 주소일 것이다.

재귀 중이라면 matrixpower 함수 안을 계속 돌며 행렬의 거듭제곱이 진행되고 재귀가 끝났으면 다시 main 함수로와 거듭제곱한 행렬의 첫번째 값(즉. n번째 피보나치 수)를 출력하고 프로그램을 종료하는 과정을 거친다.

- **multiply_power 함수**

행렬을 제곱하는 함수이다. 행렬의 값들을 로드 한 후 행렬의 곱셈 규칙에 맞춰서 곱셈하고 다시 행렬에 저장하는 과정을 거친다.

- **multiply_basic 함수**

거듭제곱하고 있는 행렬과 기본 행렬([1, 1, 1, 0])의 행렬 곱셈이 이뤄지는 함수이다. 예를 들어 홀수 번 거듭제곱하는 상황이 있을 때(n-1이 홀수 일 때) 행렬을 제곱한 후 기본행렬을 한 번 더 곱해 홀수를 맞춰야 해서 이 함수가 필요하다.

기본 행렬의 index 값들과 거듭제곱하고 있는 행령의 index 값들을 로드해서 행렬의 곱셈 규칙에 맞게 계산한 후 다시 행렬에 저장하는 과정을 거친다.

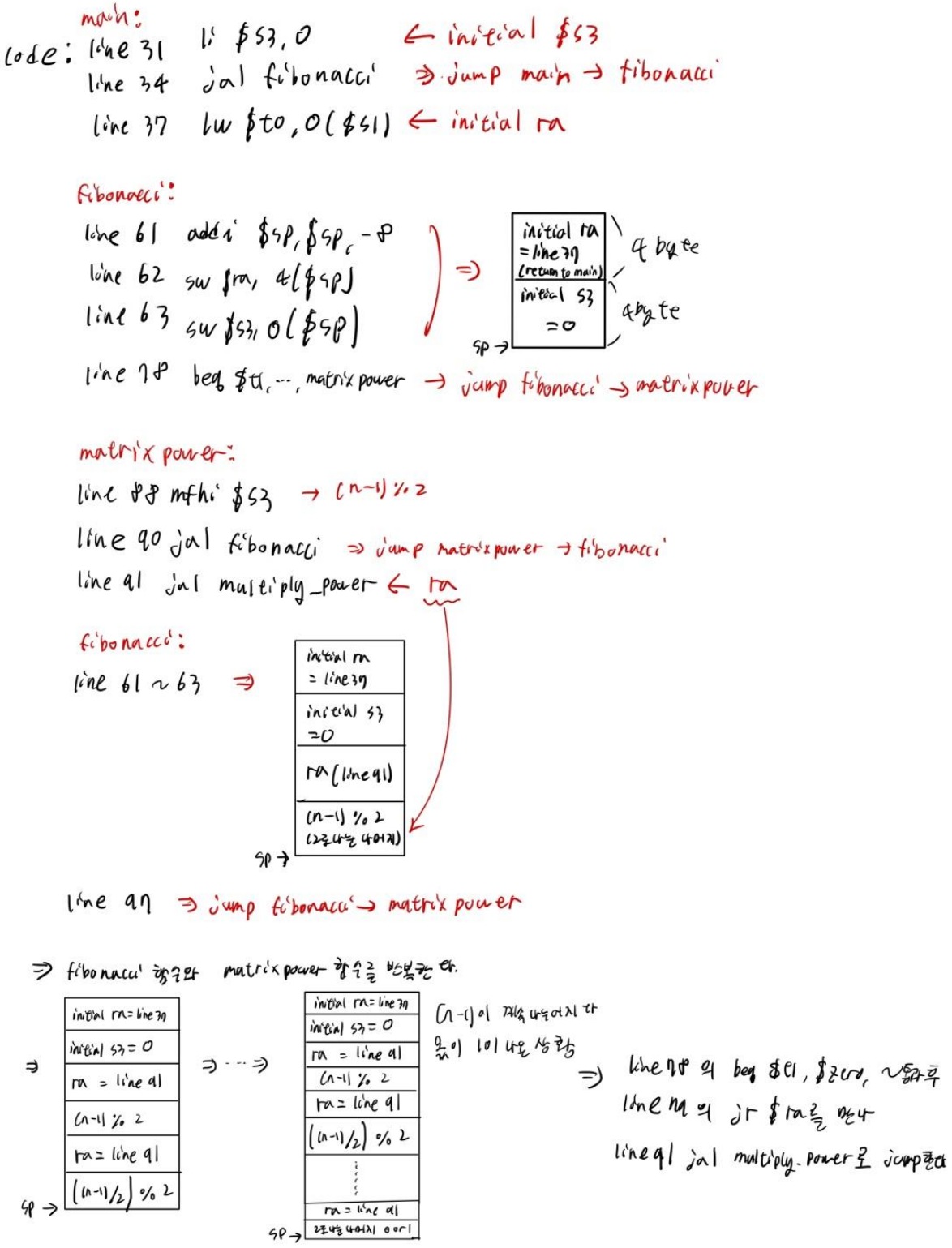
**2. 정리**

n번째 피보나치 수를 알기 위해서 행렬[1, 1, 1, 0]을 (n-1)번 거듭제곱하여 1행 1열의 index 값을 알고자 하는 프로그램이다. 이를 위해 (n-1)을 2로 반복해서 나누고 나눌 때마다 return address와 나머지를 기억하기 위해 스택에 저장하는 과정을 거친다. (n-1)을 2로 나누다가 더 이상 나눌 수 없는 1이 되었을 때, 스택에 저장되어 있던 ra와 나머지 값들을 읽어내며 다시 재귀하는 과정을 거친다.

2로 나눈 나머지가 0일 때는 multiply_power 함수를 통해 행렬을 제곱하기만 하고 나머지가 1이라면 multiply_power를 통해 제곱한 후 multiply_basic으로 한 번의 기본 행렬 곱을 추가하여 기존의 (n-1)값에 맞춰간다.

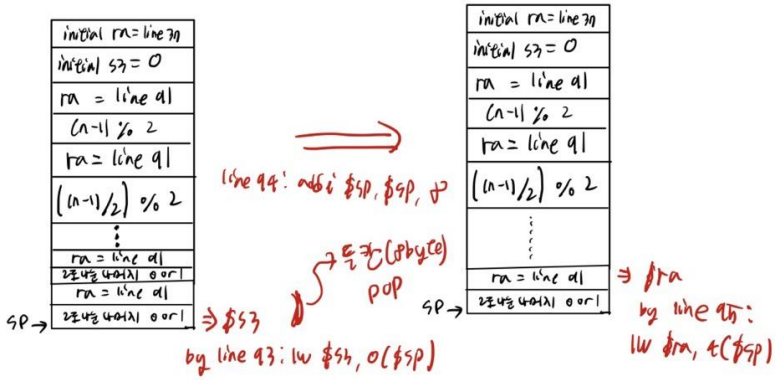
이 과정을 한 번 반복할 때 마다 스택을 두 item씩(나머지와 ra) pop한다. 마지막으로 스택에 남아 있는 제일 상단 두 item은 main 함수에서 fibonacci 함수로 넘어올 때 저장된 $s3(=0), $ra(=main 함수 'jal fibonacci'의 다음 명령어 주소 값)이다. 즉, 재귀가 끝나면 스택에 저장되어 있는 마지막 ra값을 통해 main 함수로 복귀한 후 피보나치 수를 출력하고 프로그램을 끝낸다.

### 3. Stack allocation layout



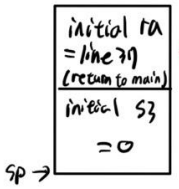
이후 matrix power 함수에서 재귀하여 stack이 pop된다.

↳ ra가 계속 line 91로 지정되어 있으므로 함수가 반복실행 됨




위 과정이 반복되다가 마지막에는 스택이 다음과 같이 남는다.

↳ matrix power function을 반복




⇒ line 94: addi $sp, $sp, 4 반복

스택의 두 아이템씩 pop 된다.

⇒ ∴ main함수로 돌아가 피보나치 수를 출력하고 종료한다.

(line 77)