GCD

소스 코드는 두 수 lhs, rhs 를 입력 받으면 euclidean algorithm 을 이용해 최대공약수를 구한다 .

```
11  gcd:
12      #------Your code starts here------
13      #LHS: a0, RHS: a1
14      addi t0,a0,0
15      addi t1,a1,0
16  Loop:
17      beq t0,t1,Exit
18      bge t1,t0,Else
19      sub t0,t0,t1
20      j Loop
21  Else:
22      sub t1,t1,t0
23      j Loop
24  Exit:
25      add a0,zero,t0
26      #Load return value to reg a0
27      #------Your code ends here------
28
29      #Ret
30      jr  ra
```

1. 함수가 호출 될 때 두 인자 lhs, rhs 는 두 레지스터 a0, a1 을 통해 전달 받고 이를 addi 를 이용해 t0, t1 에 복사시켰다 . Caller saved register 인 t0, t1 을 사용해서 스택에 저장은 하지 않는다 .

2. 이 후 소스코드의 while(lhs != rhs) 문은 어셈블리어로 변환하는 과정에서 beq 명령어를 통해 t0, t1 값이 같으면 Exit label 로 , 같지 않으면 연산이 끝난 후 다시 beq 메모리 위치로 돌아와 조건문부터 실행하는 방식으로 구현했다 .

3. while 문 안의 if(lhs > rhs) 는 소스코드와 대조를 쉽게 하기 위해서 반대 조건인 bge t1, t0, Else 를 써서 lhs 가 rhs 보다 크면 다음 명령어 실행 , lhs 가 rhs 보다 작으면 Else label 로 이동하는 방식을 쓰고 두 조건에서 각각 sub t0,t0,t1 과 sub t1,t1,t0 를 이용해 lhs = lhs - rhs 와 rhs = rhs - lhs 를 구현했다 .

4. t0 와 t1 의 값이 같아지면 return 값을 전달해주는 a0 에 값을 복사시키고 함수를 호출한 다음 명령어의 주소를 담고 있는 return address 를 이용해 main 함수로 돌아간다 .

fibonacci

```
9   fibonacci:
10      #------Your code starts here------
11      #LHS: a0, RHS: a1
12
13      addi t0,zero,1
14      sd t0,0(a0)
15      addi t1,zero,2
16      blt a1,t1,Exit
17      sd t0,8(a0)
18      addi t0,zero,2
19  Loop:
20      bge t0,a1,Exit
21      slli t1,t0,3
22      add t2,a0,t1
23      ld t3,-8(t2)
24      ld t4,-16(t2)
25      add t3,t3,t4
26      sd t3,0(t2)
27      addi t0,t0,1
28      j Loop
29  Exit:
30      #Load return value to reg a0
31      #------Your code ends here------
32
33      #Ret
34      jr  ra
```

소스 코드는 long 배열의 시작 주소와 count 를 받고 배열 끝 두 원소를 더한 값을 배열에 추가시키는 방식으로 피보나치 수의 배열을 만든 후 count 만큼 출력한다 .

1. 배열의 시작 주소 opt 와 count 값을 register a0, a1 을 통해 전달 받는다 . sd 명령어와 배열의 주소 a0, offset 을 이용해 배열의 첫 원소와 두번째 원소를 1 로 초기화한다 . 이 때 blt 를 이용해 count 가 t1 에 초기화한 상수 2 보다 작으면 Exit label 로 간다 .

2. t0 를 2 로 선언하고 소스 코드의 i 대신 사용한다 . for(int i = 2 ; i < count ; i++) 문의 반대 조건인 bge t0,a1 을 이용해 i 가 count 보다 크면 Exit label 로 가고 그렇지 않으면 loop 를 돌게 한다 .

3. slli 을 이용해 t0 의 비트를 왼쪽으로 3 칸 옮기면 t0(i) 에 8 을 곱한 값 , 배열 주소의 offset 이 나오고 이를 t1 에 저장한다 .

4. 배열의 시작 주소 (a0) 와 offset(t1) 을 더하면 배열의 i 번째 원소의 주소가 나오고 이를 t2 에 저장한다 .

5. opt[i-1] 와 opt[i-2] 를 ld 명령어를 통해 t3,t4 에 저장 후 더한 값을 sd 를 이용해 t2 가 주소를 저장하고 있는 opt[i] 에 저장한다 .

6. 배열의 시작 주소 (a0) 는 변화가 없으므로 바로 jr ra 를 통해 main 함수의 다음 명령어로 돌아간다

Maze - 1

```
11  solve_maze:
12      #------Your code starts here------
13      #maze: a0, width: a1, height: a2
14      addi sp,sp,-72
15      sd  ra, 64(sp)
16      sd  s8, 56(sp)
17      sd  s1, 48(sp)
18      sd  s2, 40(sp)
19      sd  s3, 32(sp)
20      sd  s4, 24(sp)
21      sd  s5, 16(sp)
22      sd  s6, 8(sp)
23      sd  s7, 0(sp)
24      addi s8,zero,20
25      addi s1,zero,0
26      addi s2,zero,1
27      addi s3,zero,2
28      addi s4,zero,3
29      addi s5,a0,0
30      addi s6,a1,0
31      addi s7,a2,0
32      addi a0,zero,0
33      addi a1,zero,0
34      addi a2,zero,0
35      addi a3,zero,2
36      jal ra, your_funct
37      ld  s7, 0(sp)
38      ld  s6, 8(sp)
39      ld  s5, 16(sp)
40      ld  s4, 24(sp)
41      ld  s3, 32(sp)
42      ld  s2, 40(sp)
43      ld  s1, 48(sp)
44      ld  s8, 56(sp)
45      ld  ra, 64(sp)
46      addi sp,sp,72
47
48      #Load return value to reg a0
49      #------Your code ends here------
50
51      #Ret
52      jr ra
```

소스코드의 traverse 함수는 배열의 각 위치에서 각 위치가 배열을 벗어나는지 , 길이 있는 곳인지 체크 후 위쪽 , 왼쪽 , 오른쪽 , 아래쪽 원소에서 traverse 를 재귀적으로 실행하는 함수이다 .

1. main function 으로 돌아갈 때 saved register 와 return address register 가 유지되어야 하기 때문에 앞으로 사용할 s1~s8, ra 를 저장한다 .

2. solve_maze 함수가 호출되면 #define 으로 정의된 MAX_DEPTH, T_UP, T_LEFT, T_RIGHT, T_DOWN 과 static 으로 정의된 *g_maze, g_width, g_height 를 각각 s8,s1~s7 에 저장하고 traverse 에 인자로 줄 a0~a3 를 0,0,0,2 로 초기화한다 . 또 return address 에 37 번째 줄 명령어의 주소를 저장하고 your_funct 함수를 호출한다 .

3. 모든 노드를 DFS 방식으로 돌고 your funct 가 끝나면 저장된 s1~s8, ra 를 로드하고 main function 으로 돌아간다 .

Maze - 2

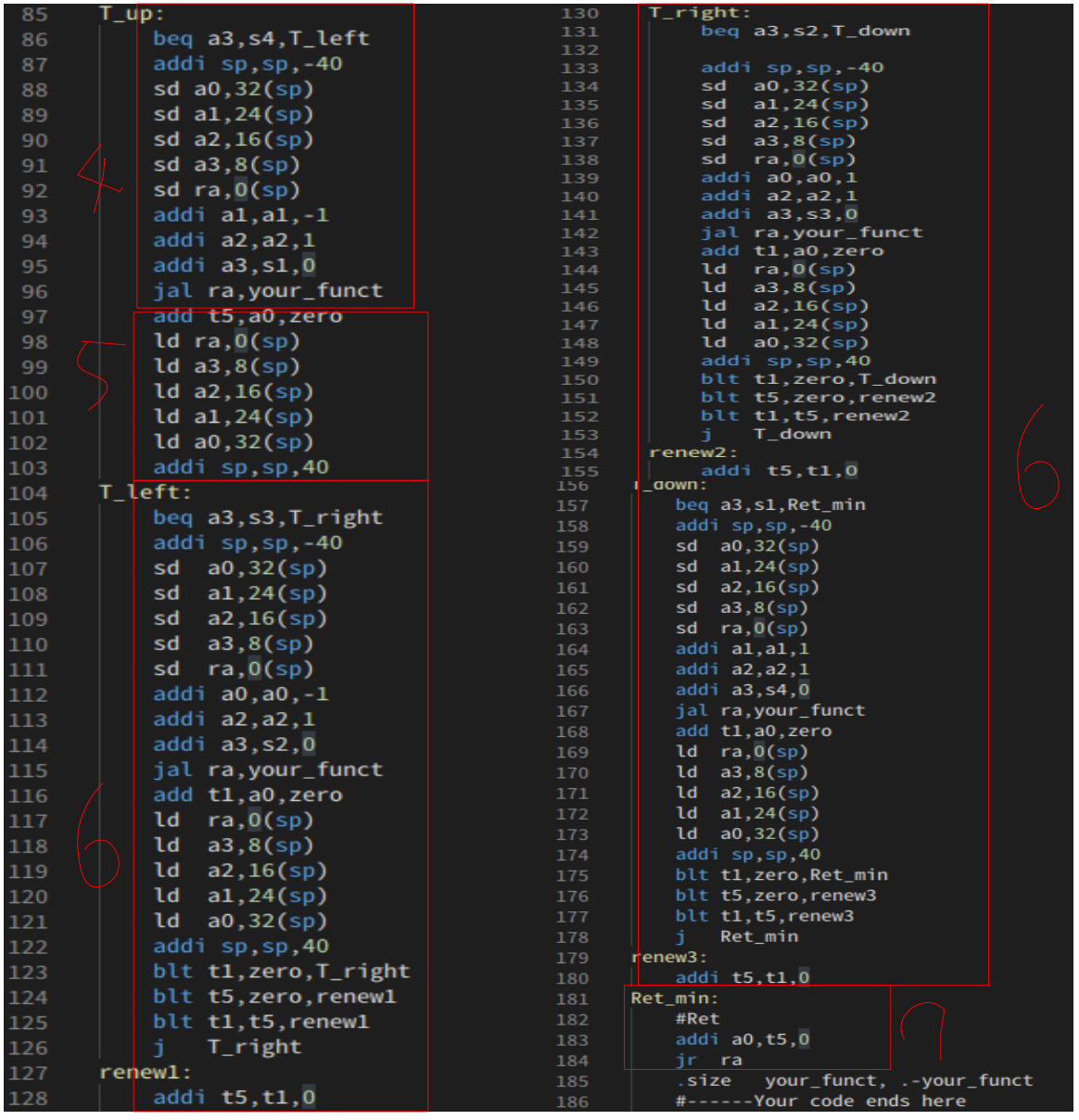
```
61  your_funct:
62      bge a2,s8,Noway
63      blt a0,zero,Noway
64      blt a1,zero,Noway
65      bge a0,s6,Noway
66      bge a1,s7,Noway
67      mul t0,a1,s6
68      add t0,t0,a0
69      slli t1,t0,3
70      add t1,t1,s5
71      ld t2,0(t1)
72      bne t2,zero,Noway
73      addi t3,s6,-1
74      addi t4,s7,-1
75      bne t3,a0,continue
76      bne t4,a1,continue
77  success:
78      addi a0,a2,0
79      jr ra
80  Noway:
81      addi a0,zero,-1
82      jr ra
83  continue:
84      addi t5,zero,-1
```

1. maze.c 에 있는 것과 같이 길이가 20 을 넘어가거나 배열을 범위를 벗어나거나 현재 노드의 값이 1 일 경우 -1 을 리턴 시켜야 한다 . 소스코드의 if(depth >= MAX_DEPTH) if(x_pos < 0 || y_pos < 0) if(x_pos >= g_width || y_pos >= g_height) 세 조건을 bge a2,s8,Noway blt a0,zero,Noway blt a1,zero,Noway bge a0,s6,Noway bge a1,s7,Noway 의 다섯 조건으로 변환 후 각각의 조건을 모두 만족시키지 않아야 다음 줄로 진행하고 하나라도 만족시키면 Noway 로 이동 후 -1 을 반환한다 .

2. if(g_maze[y_pos * g_width + x_pos]) 조건을 확인하기 위해 y_pos * g_width + x_pos 을 계산 후 t0 에 저장하고 slli 으로 3 비트 이동시켜 t1 에 저장한다 . 이는 t0 에 8 을 곱한 것과 같다 . 또 t1 에 g_maze 의 첫주소를 더한 주소에 존재하는 값을 t2 에 로드한다 . 즉 t2 는 g_maze[y_pos * g_width + x_pos] 의 값을 가지고 t2 가 1 이면 Noway 로 이동 후 -1 을 반환한다 .

3. if(x_pos == g_width - 1 && y_pos == g_height - 1) 이면 depth 를 반환해야 한다 . && 을 구현하기 보다 or 을 구현하기가 더 쉬워 위 조건의 반대 조건 bne t3,a0,continue bne t4,a1,continue 을 이용해 t3(g_width-1)==t0(x_pos) 이고 t4(g_height)==a1(y_pos) 일 경우에만 success label 로 가서 depth 를 반환하고 그렇지 않으면 continue label 로 가서 아래 코드를 진행한다 .

Maze - 3



```
85   T_up:
86       beq a3,s4,T_left
87       addi sp,sp,-40
88       sd a0,32(sp)
89       sd a1,24(sp)
90       sd a2,16(sp)
91       sd a3,8(sp)
92       sd ra,0(sp)
93       addi a1,a1,-1
94       addi a2,a2,1
95       addi a3,s1,0
96       jal ra,your_funct
97       add t5,a0,zero
98       ld ra,0(sp)
99       ld a3,8(sp)
100      ld a2,16(sp)
101      ld a1,24(sp)
102      ld a0,32(sp)
103      addi sp,sp,40
104  T_left:
105      beq a3,s3,T_right
106      addi sp,sp,-40
107      sd  a0,32(sp)
108      sd  a1,24(sp)
109      sd  a2,16(sp)
110      sd  a3,8(sp)
111      sd  ra,0(sp)
112      addi a0,a0,-1
113      addi a2,a2,1
114      addi a3,s2,0
115      jal ra,your_funct
116      add t1,a0,zero
117      ld  ra,0(sp)
118      ld  a3,8(sp)
119      ld  a2,16(sp)
120      ld  a1,24(sp)
121      ld  a0,32(sp)
122      addi sp,sp,40
123      blt t1,zero,T_right
124      blt t5,zero,renew1
125      blt t1,t5,renew1
126      j   T_right
127  renew1:
128      addi t5,t1,0
130  T_right:
131      beq a3,s2,T_down
132
133      addi sp,sp,-40
134      sd  a0,32(sp)
135      sd  a1,24(sp)
136      sd  a2,16(sp)
137      sd  a3,8(sp)
138      sd  ra,0(sp)
139      addi a0,a0,1
140      addi a2,a2,1
141      addi a3,s3,0
142      jal ra,your_funct
143      add t1,a0,zero
144      ld  ra,0(sp)
145      ld  a3,8(sp)
146      ld  a2,16(sp)
147      ld  a1,24(sp)
148      ld  a0,32(sp)
149      addi sp,sp,40
150      blt t1,zero,T_down
151      blt t5,zero,renew2
152      blt t1,t5,renew2
153      j   T_down
154  renew2:
155      addi t5,t1,0
156  T_down:
157      beq a3,s1,Ret_min
158      addi sp,sp,-40
159      sd  a0,32(sp)
160      sd  a1,24(sp)
161      sd  a2,16(sp)
162      sd  a3,8(sp)
163      sd  ra,0(sp)
164      addi a1,a1,1
165      addi a2,a2,1
166      addi a3,s4,0
167      jal ra,your_funct
168      add t1,a0,zero
169      ld  ra,0(sp)
170      ld  a3,8(sp)
171      ld  a2,16(sp)
172      ld  a1,24(sp)
173      ld  a0,32(sp)
174      addi sp,sp,40
175      blt t1,zero,Ret_min
176      blt t5,zero,renew3
177      blt t1,t5,renew3
178      j   Ret_min
179  renew3:
180      addi t5,t1,0
181  Ret_min:
182      #Ret
183      addi a0,t5,0
184      jr  ra
185      .size   your_funct, .-your_funct
186      #------Your code ends here
```

4. T_up 은 현재 위치의 위쪽 노드로 이동해 traverse 를 진행하는 함수이다 . 따라서 prev_trav(a3) 가 T_DOWN(s4) 이면 T_up 을 break 하고 T_left 로 이동한다 . 그렇지 않으면 a0~a3,ra 를 저장한 후 a0~a3 를 위쪽 노드에 맞게 초기화 한 후 traverse(your_funct) 를 새로 시작한다 .

5. your_funct 의 반환값을 min(t5) 에 저장 후 a0~a3,ra 를 다시 복구시킨다 .

6. T_left, T_right, T_down 도 T_up 과 같은 방식이나 traverse 반환값을 result(t1) 에 저장 후 min 과 비교해 result 가 더 작다면 renew label 로 이동 후 result 값을 min(t5) 에 저장한다 .

7. T_up~T_down 이 모두 종료되면 min(t1) 값을 a0 에 저장 후 반환한다 .