

시스템프로그래밍 PA2 보고서

2017312848 박종욱

1. gcd와 lcm 찾기

1) gcd

```
int gcd (int n, int m, int l)
{
    int i, j;

    if (n > m) j = n;
    else      j = m;

    if (j > 1) j = 1;

    for (i=j; i>0; i--)
        if (n%i == 0 && m%i == 0 && l%i == 0)
            break;
    return i;
}
```

93	push %rbx	112	.L2 :	129	push %rdx
94	push %rcx	113	push %rdx	130	movq \$0, %rdx
95		114	movq \$0, %rdx	131	movq %rdx, %rax
96	movq \$0, %rbx	115	movq %rdi, %rax	132	idivl %ebx
97	movq \$0, %rcx	116	idivl %ebx	133	cmpl \$0, %edx
98		117	cmpl \$0, %edx	134	pop %rdx
99	cmpl %esi, %edi	118	pop %rdx	135	je .L1
100	movl %esi, %ecx	119	jne .L3	136	
101	cmovg %edi, %ecx	120		137	.L3 :
102		121	push %rdx	138	decl %ebx
103	cmpl %edx, %ecx	122	movq \$0, %rdx	139	cmpl \$0, %ebx
104	cmovg %edx, %ecx	123	movq %rsi, %rax	140	jg .L2
105	movl %ecx, %ebx	124	idivl %ebx	141	
106		125	cmpl \$0, %edx	142	.L1:
107		126	pop %rdx	143	movq %rbx, %rax
108	cmpl \$0, %ebx	127	jne .L3	144	pop %rcx
109	jle .L1			145	pop %rbx

rbx와 rcx를 사용할 것이고

esi, edi, edx. 각각에 들어있는 n,m,l의 if else문을 다음과 같이 나타냈습니다.

또한 수업시간에 배운 For Version을 Goto Version으로 바꾸는 방법을 사용해서

처음에 Init 실행후 Test를 확인한 다음에 .L2로 루프가 시작되며

연속된 if문을 세 가지로 나눈 뒤 각각 실패하면 바로 다음 포문을 실행하게 해주는 L3로 이동하고 만약 모두 만족하여 수행을 하면 .L1을 이용해서 무한루프를 탈출하게 된 뒤 rax로 최종값을 반환해줍니다.

2) lcm

```
int lcm (int n, int m, int l)
{
    int i, j;

    if (n > m)    j = n;
    else         j = m;

    if (j > l)    j = l;

    for (i=j; i <= m*n*l; i+=j)
        if (i%n==0 && i%m==0 && i%l == 0)
            break;
    return i;
}
```

```

151 push %r10
152 push %rbx
153 push %rcx
154
155 movq $0, %r10
156 movq $0, %rbx
157 movq $0, %rcx
158
159 cmpl %esi, %edi
160 movl %esi, %ecx
161 cmovg %edi, %ecx
162
163 cmpl %edx, %ecx
164 cmovg %edx, %ecx
165
166
167 movq %rdi, %r10
168 imulq %rsi, %r10
169 imulq %rdx, %r10
170
171 movl %ecx, %ebx
172 cmpq %r10, %rbx
173 jg .L4
174
175 .L5 :
176 push %rdx
177 movq $0, %rdx
178 movq %rbx, %rax
179 idivl %edi
180 cmpl $0, %edx
181 pop %rdx
182 jne .L6

184 push %rdx
185 movq $0, %rdx
186 movq %rbx, %rax
187 idivl %esi
188 cmpl $0, %edx
189 pop %rdx
190 jne .L6
191
192 push %rsi
193 movq %rdx, %rsi
194 push %rdx
195 movq $0, %rdx
196 movq %rbx, %rax
197 idivl %esi
198 cmpl $0, %edx
199 pop %rdx
200 pop %rsi
201 je .L4
202
203 .L6 :
204 addq %rcx, %rbx
205 cmpq %r10, %rbx
206 jle .L5
207
208 .L4 :
209 movq %rbx, %rax
210 pop %rcx
211 pop %rbx
212 pop %r10

```

이번에는 r10, rbx, rcx를 사용하였습니다.

앞부분은 gcd와 같았습니다. 하지만, 포문의 INIT과정과 TEST과정을 지낸후

역시나 마찬가지로 이프문을 나누어서 브레이크 시에는 바로 L4로 가게 해주었고 이프문이 충족되지 않는다면 L6에서 테스트를 하여 무한 루프를 실행했습니다.

gcd와의 큰 차이점은 이프문에서 나누는 수와 나눔을 당하는 수가 바뀐 점 입니다.

2. factorial

```
int factorial (int n)
{
    if (n==0 || n==1) return 1;
    return n * factorial(n-1);
}

    cmp1 $0, %edi
    je .L1
    cmp1 $1, %edi
    je .L1

    push %rdi
    decq %rdi
    call _factorial
    pop %rdi

.L2 :
    imull %edi, %eax
    ret

.L1 :
    movl $1, %eax
    ret
```

edi가 0일때 1일때 1을 리턴하여 바로 루프를 끝내게 해주었습니다.

만약 이프문이 충족이 안된다면

n을 저장한 후에 n의 1을 뺀 수를 다시 재귀시켰고 그 후 factorial(n-1)이 들어있는 값 eax와 다시 1을 빼기전 저장한것을 꺼내온 edi와 곱셈을 해주었습니다.

3. maze

```

/* data section start */
.data
/* ===== Your code can be here ===== */
INF:      .int 0x190
MAX_INDEX: .int 80
/* ===== */

/* code section start */
.text
.globl findPath
findPath:
/* ===== Start of your code ===== */
push %rbp
push %r10
push %r12
push %r13
push %r11
push %r15
push %rbx

movq $0, %r10
movq $400, %r10
movq $1, %r12

movq $0, %r15 #index
movl %edx, %r15d
imull %ecx, %r15d
leal (%r15d, %esi, 1), %r15d

push %r14 #is it end point?
movq $0, %r14
movl %ecx, %r14d
imull %r8d, %r14d
subl $0x1, %r14d
cmpl %r14d, %r15d
jne .L1 #skip
movl (%r9, %r15, 1), %r14d
cmpl $0, %r14d
movl $INF, %eax
cmovle %edi, %eax
jmp .L2 #finsh
.L1 :

```

data에는 400인 INF와 배열의 끝인 MAX_INDEX를 정의했습니다.

또한 findPath 코드에서는 rbp~rbx까지 다양한 레지스터를 사용했습니다.

main.c를 gdb로 실행하여 차례대로 인풋값의 레지스터를 찾아주었습니다.

이 레지스터를 나열하여 정보를 표시해보면 다음과 같습니다.

General-purpose registers			
	63	31	0
RAX			EAX
RBX			EBX
RCX			ECX
RDX			EDX
RSI			ESI
RDI			EDI
RBP			EBP
RSP			ESP
R8			R8D
R9			R9D
R10			R10D
R11			R11D
R12			R12D
R13			R13D
R14			R14D
R15			R15D
RIP			EIP
RFLAGS			EFLAGS

register naming

#is it end point를 보면

r15d를 이용해서 index를 표현하였고 $x + y * w$ 를 r14d로 계산하여 비교했습니다.

m[index]는 m의 시작주소 r9에 index를 더해서 r14d에 임시저장한 후에 test를 해주었습니다. skip과 finish는 이 이프문의 이동결과입니다. (주석참고)

```

59 movb $0x2, (%r9, %r15, 1)  #m[index] = 2;
60
61
62 movq $0, %r11      #state 0
63 movl %esi, %ebx
64 movq $0, %r14
65 movl %ecx, %r14d
66 subl $0x1, %r14d
67 movq $0, %r13
68 movl %edx, %r13d
69 imull %ecx, %r13d
70 leal 1(%r13d, %esi, 1), %r13d
71 jmp .L3
72 .L6 :
73 push %rdi
74 push %rsi
75 addq $0x1, %rdi
76 addq $0x1, %rsi
77 callq findPath
78 pop %rsi
79 pop %rdi
80 jmp .L8
162 .L3 :      #if
163 cmpl %ebx, %r14d
164 jle .L10
165 cmpl $0, %r13d
166 jl .L10
167 cmpl $MAX_INDEX, %r13d
168 jg .L10
169 movzbq (%r9, %r13, 1), %r13
170 cmpl $0, %r13d
171 jne .L10
172
173 cmpl $1, %r11d
174 jl .L6 #state 0's normal
175 je .L7 #state 1's normal
176 cmpl $2, %r11d
177 je .L11 #state 2's normal
178 jg .L13 #state 3's normal

```

```

if((x < w - 1) && !m[right]){
    temp = findPath(1 + 1, x + 1, y, w, d, m);
    total_length = min(temp, total_length);
    is_blocked = FALSE;
}

```

다시 돌아오는것을 방지하기위해 해당 미로의 값을 2로 마킹해놓았습니다.

state 0는 처음 이프문으로 state는 r11에 0으로 저장해두고

ebx에 x값, r14d에 비교할 w-1값을 저장

r13d값에 right값을저장한 후에 L3으로 이동합니다.

L3에서는 이프문을 확인하며 충족이 안되면 L10으로 이동합니다.

모두 충족이 될 시에는 각 스테이트에 맞는 곳으로 이동시킵니다. (계산하러감)

먼저 L10입니다.

```

153 .L10 : #checking start point
154 cmpl $1, %r11d
155 jl .L5 #state 1's start
156 je .L9 #state 2's start
157 cmpl $2, %r11d
158 je .L12 #state 3's start
159 jg .L4

```

해당 이프문에 충족이 안되었기에 스테이트를 따져 다음 이프문으로 바로 이동시킵니다.

다음은 충족된 루프문이고 각 루프문은 해당 스테이트 아래에 마킹해두었습니다.

첫 번째 그림에서 72번째 줄을 보면 state 0의 실행지가 있습니다. 이곳에서 1과 x를 저장후 1씩 더하여 재귀시킨후 다시 팝하여 L8로 이동합니다.

```

147 .L8 :
148     cmpl %eax, %r10d
149     cmovg %eax, %r10d
150     movl $0, %r12d
151     jmp .L10

```

L8에서는 temp와 total_length를 비교후

total_length가 크면 temp로 업데이트 시켜줍니다. (고로 작은 값이 들어가짐)
이후 L10으로 이동하여 다음 스테이트로 이동시켜줍니다.

```

82 .L5 :      #state 1
83     movq $1, %r11
84     movl %edx, %ebx
85     movq $0, %r14
86     movl %r8d, %r14d
87     subl $0x1, %r14d
88     movq $0, %r13
89     movl %edx, %r13d
90     addl $1, %r13d
91     imull %ecx, %r13d
92     leal (%r13d, %esi, 1), %r13d
93     jmp .L3
94 .L7 :
95     push %rdi
96     push %rdx
97     addq $0x1, %rdi
98     addq $0x1, %rdx
99     callq findPath
100    pop %rdx
101    pop %rdi
102    jmp .L8

```

```

104 .L9 :      #state 2
105     movq $2, %r11
106     movl %esi, %r14d
107     movq $0, %rbx
108     movq $0, %r13
109     movl %edx, %r13d
110     imull %ecx, %r13d
111     leal -1(%r13d, %esi, 1), %r13d
112     jmp .L3
113 .L11 :
114     push %rdi
115     push %rsi
116     addq $0x1, %rdi
117     subq $0x1, %rsi
118     callq findPath
119     pop %rsi
120     pop %rdi
121     jmp .L8

```

```

124 .L12 :     #state 3
125     movq $3, %r11
126     movl %edx, %r14d
127     movq $0, %rbx
128     movq $0, %r13
129     movl %edx, %r13d
130     subl $1, %r13d
131     imull %ecx, %r13d
132     leal (%r13d, %esi, 1), %r13d
133     jmp .L3
134 .L13 :
135     push %rdi
136     push %rdx
137     subq $0x1, %rdx
138     addq $0x1, %rdi
139     callq findPath
140     pop %rdx
141     pop %rdi
142     jmp .L8

```

이후 다른 이프문은 이하동문입니다.

이 스테이트가 모두 끝났을 경우 L10을 다시보면

```

153 .L10 :     #checking start point
154     cmpl $1, %r11d
155     jl .L5 #state 1's start
156     je .L9 #state 2's start
157     cmpl $2, %r11d
158     je .L12 #state 3's start
159     jg .L4

```

L4로 이동함을 알 수 있습니다.


```

181  .L4 : #m[index] = 0;
182  movb $0x0, (%r9, %r15, 1)    #m[index] = 0;
183  cmpl $0, %r12d
184
185  movq $0, %rax
186  movq $400, %rax
187  cmove %r10d, %eax
188
189
190  .L2 :
191  pop %rbx
192  pop %r14
193  pop %r15
194  pop %r11
195  pop %r13
196  pop %r12
197  pop %r10
198  pop %rbp
199  /* ===== End of your code ===== */
200  ret

```

역시나 L4는

```

64      m[index] = 0;
65      if(is_blocked)
66          return INF;
67      else
68          return total_length;
69  }
70

```

c파일의 함수를 구현하였고 최종적으로 푸쉬한 데이터를 모두 팝하는 것을 보실 수 있습니다.