[SWE2001 System Program - Spring 2021]

x86 Assembly

Due: 11:59 PM, 9th May 2021

C코드로 주어진 함수를 어셈블리 언어로 구현하는 것을 통해, 어셈블리언어에 익숙해지고 컴퓨터 시스템 내부 동작을 이해한다.

2. Overview

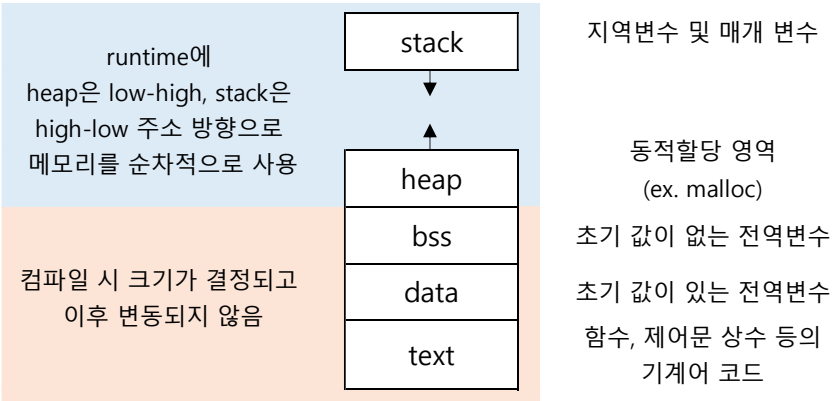
C언어로 쓰여진 코드(.c)를 바탕으로 x86 어셈블리 언어로 쓰여진 코드(.s)를 해석하고, 아래의 세 함수를 어셈블리 언어로 변환한다.

- 주어진 수들의 최대공약수, 최소공배수 찾기
- n! (팩토리얼) 계산
- 미로 최단 경로 찾기

각 항목에 대한 파일들은 서브 디렉토리에 저장되어 있다. 자세한 설명은 4장을 참고한다.

3. Assembly coding basics

3.1 어셈블리 코딩과 메모리 세그먼트



프로그램을 만들기 위해서는, C언어나 기타 언어를 이용해 작성한 후 해당 소스를 컴파일 하게 된다. 그렇게 컴파일 된 프로그램의 정보는 데이터의 종류에 따라 메모리 상에 다른 곳에 위치하게 된다.

프로그램의 실행 시 할당, 해제되는 stack(push, pop)과 heap(ex. malloc)영역과 달리, text, bss, data영역은 프로그램 컴파일 시 크기, 위치가 결정된다. 어셈블리 언어를 이용해 프로그램 소스를 작성 시, 세 영역(text, bss, data)의 시작을 명시적으로 선언해야 한다.

```
.data
    /* 초기화 된 전역 변수의 선언 */
.bss
  /* 초기화 되지 않은 전역 변수의 선언 */
.text
    /* 기계어 코드 */
```

3.1.1 data 영역

이번 과제에서는 bss영역은 사용하지 않고, data영역에서의 변수 선언을 이용하도록 한다. data영역은 초기화된 전역 변수를 저장하는 공간으로, 아래와 같은 선언 형태를 갖는다.

```
.data
        /* [Symbol명]: [변수 타입] [초기값] */
```

data영역에서 사용되는 변수 타입은 아래와 같다.

.byte	1-byte 값
.ascii	텍스트 string
.asciz | .string	NULL값으로 끝나는 텍스트 string
.short | .hword	2-byte 크기의 정수
.int | .word	4-byte 크기의 정수
.long	8-byte 크기의 정수
.float | .single	Single-precision floating-point
.double	Double-precision floating-point

+) space 타입의 사용

.space	1-byte 단위의 공간

다른 변수 타입들과 달리 .space는 초기값이 아닌 할당할 공간의 크기를 인자로 받는다.

```
.data
 buffer:    .space   8
```

좌측과 같이 선언할 경우 buffer라는 이름의 8 bytes 메모리 공간이 할당된다.

- 배열의 사용 -

x86 어셈블리 프로그래밍에서는 아래와 같이 하나의 심볼에 같은 타입의 변수 여러 개를 선언하는 것이 가능하다. 연속하게 선언되는 변수는 실제 메모리 상에 연속하게 배치된다.

```
.data
array: .long 100,150,200,250
```

long은 8byte크기의 정수를 의미한다. 따라서, "array+16" 위치의 메모리 주소 접근을 통해 array라는 데이터의 3번째 값인 200을 참조할 수 있다

3.1.2 text영역

함수 명에 대응되는 Symbol을 선언하고 이후 기계어 코드를 작성한다. 기계어 코드 작성법은 6-8주차 수업자료를 참고한다.

```
.text
/* [Symbol명]:                  */
/*          기계어 코드          */
/*            ...               */
```

- System Call -

시스템 콜은 응용 프로그램이 운영체제 커널이 제공하는 서비스에 접근하기 위해 사용되는 인터페이스이다. 시스템 콜은 여러 종류의 기능으로 나뉘어져 있다 (프로세스제어, 파일조작, 장치조작, 통신 및 보호 등). 각 시스템 콜에는 고유 번호가 할당된다. 커널은 시스템 콜이 호출되면 고유 번호에 따라 지정된 동작을 수행한다.

어셈블리 어에서는 "`syscall`"을 이용해 시스템 콜을 호출한다. "`syscall`" 호출 전 %rax에 시스템콜 번호를 명시한다. 매개변수가 필요한 경우 <%rdi, %rsi, %rdx, %rcx, %r8 , %r9>의 general purpose register가 사용된다. 시스템 콜의 반환값은 %rax에 저장된다.

Syscall #	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
%rax	%rdi	%rsi	%rdx	%rcx	%r8	%r9

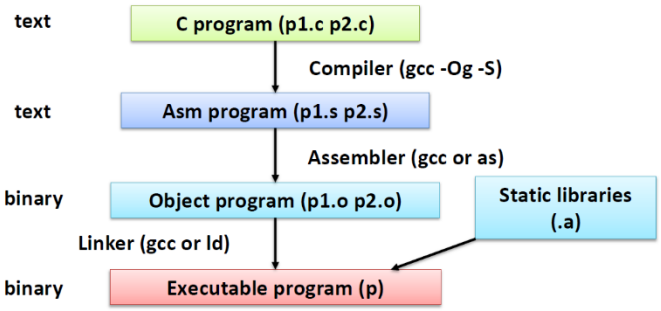
Return value
%rax

아래는 대표적인 시스템콜들의 레지스터 표이다.

%rax	System call	%rdi	%rsi	%rdx	%rcx	%r8	%r9
0	sys_read	unsigned int **fd**	char ***buf**	size_t **count**			
1	sys_write	unsigned int **fd**	const char ***buf**	size_t **count**			
2	sys_open	const char ***filename**	int **flags**	int **mode**			
3	sys_close	unsigned int **fd**					
57	sys_fork						
60	sys_exit	int **error_code**					

3.3 Global Symbol

앞서 data 영역, text 영역에서 선언된 Symbol들은 기본적으로 해당 파일 내에서만 참조 및 사용될 수 있다. 즉, 링커는 각 파일 내의 Symbol정보를 알 수 없다. 링커에게 Symbol의 정보를 전달하고자 하는 경우 ".globl [Symbol명]"를 사용한다. 링커는 전달된 Symbol 정보를 바탕으로 다른 파일에서 참조된 심볼을 연결해준다.



전체 프로그램의 시작 지점인 main은 .globl 로 선언한다.

```
.globl main

   ...

.text
main:
```

3.4 예제 코드

C코드를 어셈블리어로 변환하는 예제 코드를 제공한다. (example.c, example.s)
example파일은 두 수를 비교하고 조건에 따라 메시지를 출력한다.

< example의 c코드 >

```c
#include <unistd.h>

int main (void)
{
        int i = 2, j = 3;

        if(i<j) write(1, "bye_world\n", 10);
        else    write(1, "hello_world\n", 12);

        return 0;
}
```

< assembly 코드 >

```
  .globl main

  .data
  msg_0: .string "hello_world\n"
  msg_1: .string "bye_world\n"
  len_0 = 12
  len_1 = 10

  .text
  main:
  movl $2, %ebx              # i = 2
  movl $3, %ecx              # j = 3
  cmpl %ebx, %ecx            # === if (i <j)
  jle .L1                    #  goto L1

  movl $1, %eax              # write (            | syscall #: 1
  movl $1, %edi              #  1,                | fd (stdout:1)
  movl $msg_1, %esi          #  "bye_world\n",    | buffer address
  movl $len_1, %edx          #  10                | buffer length
  syscall                    # );
  jmp .L2

  .L1:                       # === else
  movl $1, %eax              # write (            | syscall #: 1
  movl $1, %edi              #  1,                | fd (stdout:1)
  movl $msg_0, %esi          #  "hello_world\n",  | buffer address
  movl $len_0, %edx          #  12                | buffer length
  syscall                    # );

  .L2:
  movl $60, %eax             # exit (             | syscall #: 60
  movl $0, %ebx              #  0                 | error_code
  syscall                    # );
```

4. Details

제공되는 디렉토리에는 세 개의 서브 디렉토리가 존재하며, 각 문항에 관련된 파일이 담겨있다. (./pa2-1 최대공약수, 최소공배수, ./pa2-2 팩토리얼, ./pa2-3 미로찾기). 문항 별로 아래와 같은 파일이 제공된다.

- main함수 및 어셈블리어로 변환해야 하는 함수가 담긴 C코드 (.c)
  - 어셈블리어로의 변환을 위한 참고 파일이다.
- 주어진 C코드에 대응되는 어셈블리 파일 (.s)
  - 비워진 부분을 구현하여 제출한다.
- Makefile
  - 각 항목의 실행파일을 생성 및 삭제한다.
  - make을 실행할 경우 각 항목별로 _c와 _s로 끝나는 실행파일이 생성된다. _c는 주어진 C코드로부터 생성된 실행파일로 주어진 C코드의 동작을 확인할 수 있다. _s는 assembly 코드로부터 생성된 실행 파일로 작성한 어셈블리 코드의 동작을 확인한다.

```
jg@jg-host:~/pa2_skeleton/pa2-1$ make
gcc gl.c -o gl_c
gcc -g -no-pie gl.s -o gl_s
```

- Input file
  - 각 문항별로 input파일의 이름은 고정되어 있다.
    - ◆ 최대공약수, 최소공배수 pa2-1.in / 팩토리얼 pa2-2.in / 미로찾기 pa2-3.in
  - "최대공약수, 최소공배수"와 "팩토리얼"문항에 대해서 input generator를 이용해 input file을 생성하여 확인한다. (input generator를 이용하지 않으면 오류 발생)

```
jg@jg-host:~/pa2_skeleton$ ./input_generator
Select input file to generate <1/2>?: 1
Type three integers: 6 15 30
* [6/15/30] are successfully written to [./pa2-1/pa2-1.in]
jg@jg-host:~/pa2_skeleton$ cd pa2-1
jg@jg-host:~/pa2_skeleton/pa2-1$ ./gl_s
Greate Common Divisor   [6, 15, 30] = 3
Least Common Multiple   [6, 15, 30] = 30
```

4.1 최대공약수(Greatest Common Divisor) 최소공배수(Least Common Multiple) 찾기

주어진 두 수의 최대 공배수와 최소 공약수를 반환한다. 입력으로 주어지는 두 argument "n, m"은 양의 정수이다.

< 변환해야하는 c코드 >

```c
int gcd (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>0; i--)
        if (n%i == 0 && m%i == 0 && l%i == 0)
            break;
    return i;
}
```

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

< 디렉토리 **./pa2-1** 구성>

gl.c – **main** 및 **gcd**, **lcm**함수가 들어있는 C파일
gl.s – **main** 함수는 어셈블리어로 구현되어 있으며, 비워진 **gcd**, **lcm**함수를 구현하여 제출
pa2-1.in – integer 3개를 포함한 input file 파일
Makefile – 실행파일의 생성 및 삭제

4.2 팩토리얼 수 찾기

n번째 팩토리얼 수를 찾아 반환한다. 입력으로 주어지는 argument는 양의 정수이다.

< 변환해야하는 c코드 >

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

    return n * factorial(n-1);
}
```

< 디렉토리 **./pa2-2** 구성>

factorial.c – **main** 및 **factorial**함수가 들어있는 C파일
factorial.s – **main** 함수는 어셈블리어로 구현되어 있으며, 비워진 **factorial** 함수를 구현하여 제출
pa2-2.in – integer 1개를 포함한 input file 파일
Makefile – 실행파일의 생성 및 삭제

4.3 미로 최단 경로 찾기

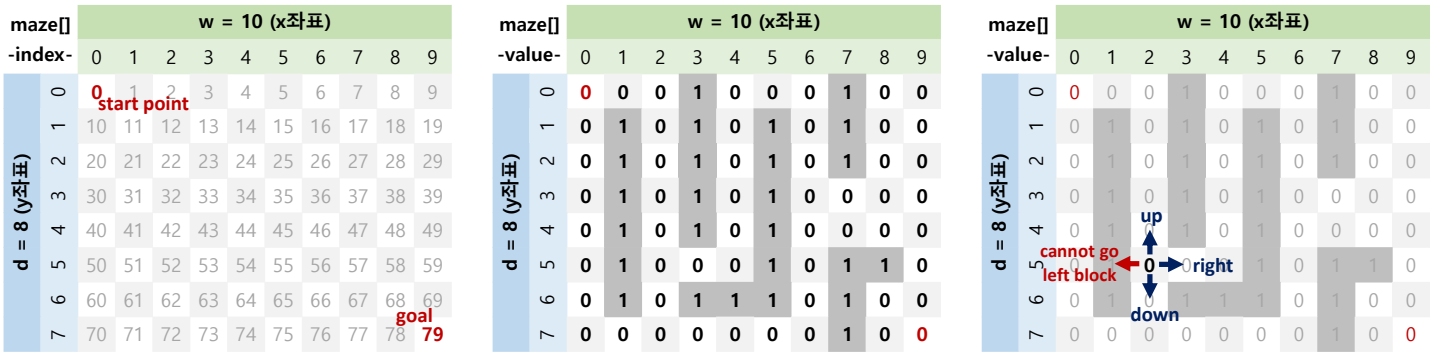
주어진 입력에 대해 입구에서 출구까지 가장 짧은 경로의 길이를 반환한다.

```
1 0001000100
2 0101010100
3 0101010100
4 0101010000
5 0101010000
6 0100010110
7 0101110100
8 0000000100
~
pa2-3.in
```

그림1. 미로 입력

입력으로 주어지는 input의 예는 좌측과 같다, 1은 지나갈 수 없는 벽(Wall)을 의미하고, 0은 지나갈 수 있는 길(Path)을 의미한다. 주어진 input은 maze라는 이름의 1차원 array(그림 2.a)에 저장된다. 좌측의 input file에대해 미로 map의 너비(w: width)는 10, map의 깊이(d: depth)는 8로 변환된다.

미로의 출발 지점은 좌측 상단(index 0)이며, 미로의 도착 지점은 우측하단이다. 즉, 그림 1과 같은 input이 주어진 경우 도착 지점은 index 79이다.
미로 내의 이동은 다음과 같이 표현된다(그림 2.c).



(a) maze 배열의 index (b) 그림1 입력에 대한 미로의 구성 (c) 미로 내의 이동 방향

그림 2. maze 배열

< 변환해야하는 c 코드 >

```
int findPath
(int l, int x, int y, int w, int d)
{
    int idx    =   x    +  y     * w;
    int up     =   x    + (y-1) * w;
    int down   =   x    + (y+1) * w;
    int left   = (x-1) +  y     * w;
    int right  = (x+1) +  y     * w;

    int total_len = INF;
    int temp = INF;
    int is_blocked = TRUE;

    // is it end point?
    if (idx == w * d - 1){
        if(maze[idx])
                return INF;
        else
                return l;
    }

    maze[idx] = 2;


                …
            Continued
                …
```

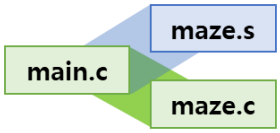
```
    // go to next point
    if ((x < w - 1) && !maze[right]) {
        temp = findPath(l+1, x+1, y, w, d);
        total_len = min(temp, total_len);
        is_blocked = FALSE;
    }
    if ((y < d - 1) && !maze[down]) {
        temp = findPath(l+1, x, y+1, w, d);
        total_len = min(temp, total_len);
        is_blocked = FALSE;
    }
    if ((x > 0) && !maze[left]) {
        temp = findPath(l+1, x-1, y, w, d);
        total_len = min(temp, total_len);
        is_blocked = FALSE;
    }
    if ((y > 0) && !maze[up]) {
        temp = findPath(l + 1, x, y - 1, w, d);
        total_len = min(temp, total_len);
        is_blocked = FALSE;
    }

    maze[idx] = 0;

    if (is_blocked)    return INF;
    else               return total_len;
}
```

< 디렉토리 ./pa2-3 구성>

gcc –no-pie maze.s main.c -o maze_s



gcc maze.c main.c -o maze_c

main.c – main 함수와 print함수가 담긴 C파일

maze.h – 심볼 링크를 위한 헤더파일

maze.c – 함수 **findPath**가 들어있는 C파일

maze.s – **findPath**를 어셈블리언어로 구현할 파일

pa2-3.in – 미로 Input file

Makefile - 실행파일의 생성 및 삭제

5. Restriction

- C코드의 순서, 연산 등을 임의로 최적화하지 않고, 그대로 변환한다.
- x86 64bit 시스템에서의 실행을 가정한다.
  - %rax, %rbx등 64bit register와 그에 따른 연산 사용 가능
- x86 assembly의 AT&T syntax를 사용한다.
  - %를 이용한 register표현 등 수업자료에서 사용한 syntax
- C코드를 disassemble하여 제출하는 경우 F학점을 받는다

6. 컴파일 및 프로그램 실행

심볼들에 상대주소를 할당하는 PIE (Position Independent Executable) 옵션을 끄고 컴파일 한다.

```
gcc -no-pie [assembly file name] -o [execution file name]
./[execution file name]
```

example >

```
jg@jg-host:~/pa2_skeleton/example$ gcc -no-pie example.s -o example
jg@jg-host:~/pa2_skeleton/example$ ./example
bye world
```

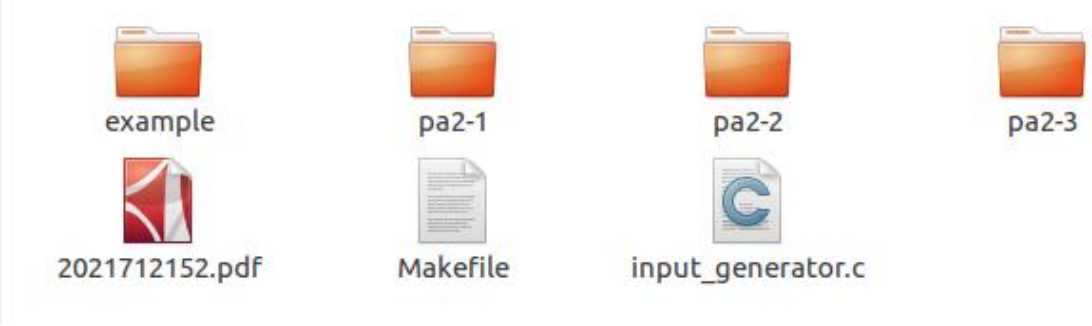
또는 제공된 make파일을 이용하여 손쉽게 컴파일 및 실행파일을 삭제할 수 있다.

```
jg@jg-host:~/pa2_skeleton/example$ make
gcc example.c -o example_c
gcc -g -no-pie example.s -o example_s
jg@jg-host:~/pa2_skeleton/example$ make clean
rm -f example_c example_s
```

또한, 상위 디렉토리에서 make 또는 make clean을 통해 하위 디렉토리의 파일들을 한번에 컴파일 할 수 있다.

7. Submission & Logistics

- 과제 수행에 대한 보고서를 작성한다. 보고서 제목은 [학번].pdf으로 한다.



- make tar를 이용해 보고서와 어셈블리 코드 파일을 한 번에 압축한다.

```
jg@jg-host:~/pa2_skeleton$ make tar STUDENT_ID=2021712152
tar -cvzf 2021712152.tar.gz pa2-1/gl.s pa2-2/factorial.s pa2-3/maze.s 2021712152
.pdf
pa2-1/gl.s
pa2-2/factorial.s
pa2-3/maze.s
2021712152.pdf
```

- 생성한 압축 파일을 canvas에 제출한다.
- 제출 양식을 지키지 않을 시 감점이 발생한다.
- 과제 제출기한 이후 매 24시간마다 20%씩 감점된다.
- 다른 사람의 과제를 copy한 경우 F 학점을 받게 된다.