서강대학교 Sogang University

# Introduction to Computer Systems

## Ex. 2 – Pen and Paper (Chapter 03)

## Chapter 3.6 : Control, Conditional Branch

Convert the following C code to your own assembly code, following the format from our lecture notes. Assume that the arguments are stored in the registers as shown in the table below.

| Register | Use(s) |
|---|---|
| %rdi | Argument x |
| %rsi | Argument y |
| %rax | Return value |

```
int my_func (int x, int y)
{
  if (x < y)
    if(2*x < y)
      return y - 2*x;
    else
      return y - x;
  else
    return y + x;
}
```

```
movq %rsi, %rax
cmpq %rsi, %rdi
jge .L2
movq %rdi, %rdx
salq $1, %rdx
cmpq %rsi, %rdx
jge .L1
subq %rax, %rdx
jmp .L3
.L1:
subq %rax, %rdi
jmp .L3
.L2:
addq %rdi, %rax
.L3:
ret
```

## Chapter 3.6 : Loops

The general form of a *for loop* in *C code* is as follows:

```
for (init-expr; test-expr; update-expr)
  body-statement
```

Also, its standard transformation into *goto code* gives:

```
init-expr;
t = test-expr;
if (!t)
  goto done;
loop:
  body-statement
  update-expr;
  t = test-expr;
  if (t)
    goto loop;
done:
```

which consists of *Inital expression*, *Initial test*, and *Body statement* with *Test expression*.
An example of a *for loop* is:

```
#define MAX 10
int func(int a)
{
  unsigned short i;
  int result = a;
  for (i = 0; i < MAX; i++){
    result += a*(i+1);
  }
  return result;
}
```

Answer the following question.

**a)** What is the value of *func(1)*?

**b)** Convert the function *func* into *goto code* version, using the format of standard transformation of *for loop*. (as suggested above)

**c)** Can we remove *Initial test* code for optimization? If so, explain a reason for it.

**a)** 56

**b)**
```
#define MAX 10
int func(int a)
{
  unsigned short i;
  int result = a;
  i = 0;
  if(!(i<MAX))
    goto done;
  loop:
    result += a*(i+1);
    i++;
    if(i<MAX)
      goto loop;
  done:
    return result;
}
```

**c)** Yes, since i = 0 is obviously less than MAX = 10, we do not need an initial test.

## Chapter 3.7 : Procedures

Assume that the following assembly code is generated for a C code, by *gcc*.

```
proc:
  pushq %rbp
  movq %rsp, %rbp
  subq $16, %rsp
  addq $-24, %rsp
  leaq -8(%rbp), %rax
  pushq %rax
  leaq -16(%rbp), %rax
  pushq %rax
  leaq -24(%rbp), %rax
  pushq %rax
  pushq $3
  (t)
  call subproc
  movq %rbp, %rsp
  popq %rbp
  ret
```

Draw a stack frame for proc before (t), and mark the location of %rsp, %rbp in it (the value of the register). Assume that the procedure *proc* starts with the following register values:

| Register | Value |
|---|---|
| %rsp | 0x800070 |
| %rbp | 0x8000F0 |

| Address | Value |
|---|---|
| 0x800068 | 0x8000F0 (%rbp) |
| 0x800060 | |
| 0x800058 | |
| 0x800050 | |
| 0x800048 | |
| 0x800040 | |
| 0x800038 | 0x800060 |
| 0x800030 | 0x800058 |
| 0x800028 | 0x800050 |
| 0x800020 | 3 (%rsp) |