Lab 3: Loops in Assembly

CS 105

February 10, 2020

Jumping

- jX instructions
  - Jump to different part of code if condition is true

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF ^ OF) & ~ZF	Greater (Signed)
jge	~(SF ^ OF)	Greater or Equal (Signed)
jl	(SF ^ OF)	Less (Signed)
jle	(SF ^ OF) | ZF	Less or Equal (Signed)

Conditional Branching

Register	Use
%rdi	x
%rsi	y
%rax	return value

```
long absdiff(long x, long y){
  long result;

  if (x > y){
    result = x-y;
  } else {
    result = y-x;
  }

  return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi
    jle     .L4
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4:                    # x <= y
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

Loops

- All use conditions and jumps
  - do-while
  - while
  - for

Do-while Loops

Register	Use(s)
%rdi	Argument x
%rax	result

```
long bitcount(unsigned long x){
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

```
long bitcount(unsigned long x){
  long result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

```
   movq     $0, %rax     #  result = 0
.L2:                     #  loop:
   movq     %rdi, %rdx
   andq     $1, %rdx     #  t = x & 0x1
   addq     %rdx, %rax   #  result += t
   shrq     %rdi         #  x >>= 1
   jne      .L2          #  if (x) goto loop
   rep; ret
```

While Loops

Register	Use(s)
%rdi	Argument x
%rax	result

```
while (Condition) {
    Body
}
```

```
if (Condition){
    do {
      Body
    } while (Condition)
}
```

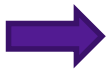
```
long bitcount(unsigned long x) {
  long result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

```
        movq    $0, %rax
        jmp     .L2
.L3:
        movq    %rdi, %rdx
        andq    $1, %rdx
        addq    %rdx, %rax
        shrq    %rdi
.L2:
        testq   %rdi, %rdi
        jne     .L3
        rep ret
```

For loops

Register	Use(s)
%rdi	Argument x
%rax	result

```
for (Init; Cond; Incr){
    Body
}
```



```
Init;
while (Cond) {
    Body;
    Incr;
}
```

```
long bitcount(unsigned long x) {
  long result;
  for (result = 0; x; x >>= 1)
    result += x & 0x1;
  return result;
}
```

```
        movq    $0, %rax
        jmp     .L2
.L3:
        movq    %rdi, %rdx
        andq    $1, %rdx
        addq    %rdx, %rax
        shrq    %rdi
.L2:
        testq   %rdi, %rdi
        jne     .L3
        rep ret
```

Initial test can often be optimized away:

```
for (j = 0; j < 99; j++)
```

Exercise: Loops

```
_loop:
  movq $0, %rax
  movq $0, %rdx
  jmp L1
L0:
  addq %rdx, %rax
  incq %rdx
L1:
  cmp %rdi, %rdx
  jl L0
  ret
```

```
long loop(long val){
  long ret = ______;
  long i;

  for(i = ____; ____; ____){

    ret = _______;

  }

  return ret;
}
```

Exercise: Loops

Register	Use(s)
%rdi	Argument **val**
%rdx	Local **i**
%rax	Local **ret**

```
_loop:
  movq $0, %rax
  movq $0, %rdx   # init
  jmp L1
L0:
  addq %rdx, %rax
  incq %rdx       # update
L1:
  cmp %rdi, %rdx # condition
  jl L0
  ret
```

```
long loop(long val){
  long ret = 0;
  long i;

  for(i = 0; i < val; i++){

    ret = ret + i;

  }

  return ret;
}
```