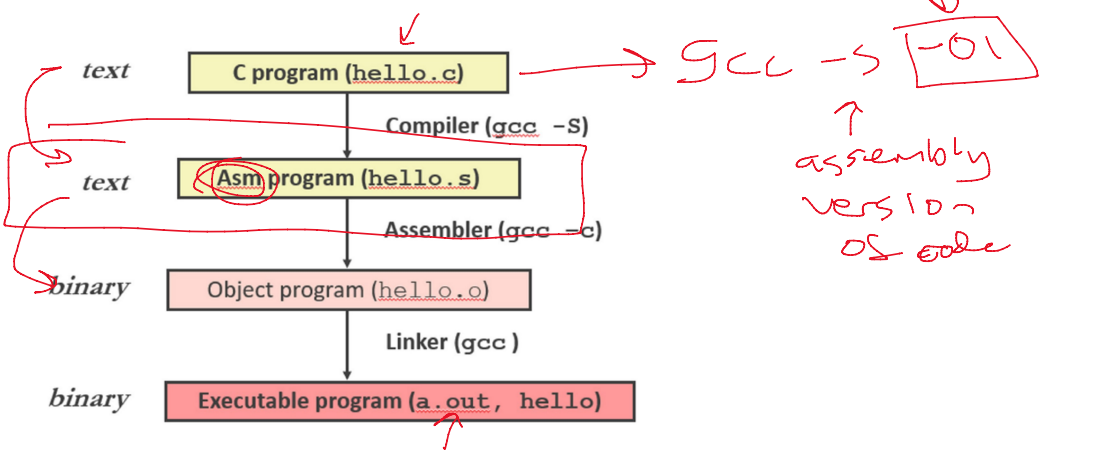
Assembly - 003

Tuesday, February 20, 2024 1:56 PM

Why learn assembly?

- optimization
  * how the compiler optimizes our code
- run-time behavior analysis
  * a programs hidden layers of abstraction
- security (reverse ensineering)
  ↳ jmp over login

Compilation Process



Machine Instruction:
on the C level:

*dest = +;

↳ on

↳ On assembly level instruction could look like this

mov %rbx, %rax

↓

t = rbx

*dest = rax

↑ look up in memory (value at that address)

↳ Machine level

↳ compact version of assembly

0x40059e: 48 89 03

mov rbx rax

↳ instruction stored as 3 bytes

Assembly Characteristics:

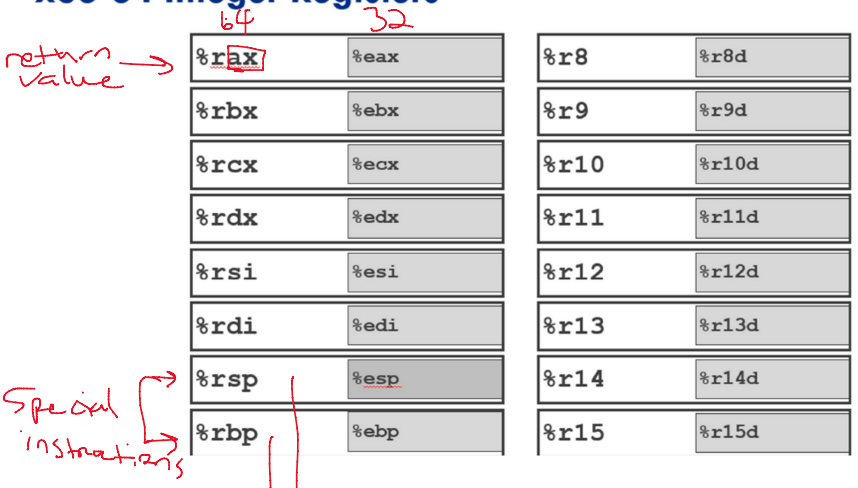
Data types:

* Integers (1, 2, 4, 8)
  ↑ short ↑ longs
  - Data values → Can store
  - address points → with ints

Floating data = 4, 8, or 16 bytes

* Byte sequences @ machine level are encoded instruction
* we don't arrays or structs
  ↳ everything stored in continously allocated bytes

x86-64 Integer Registers



- 64-bit registers are denoted with "r"
- 32-bit register are denoted with "e"
  ↳ backwards compatibility

Command notation

movq → 64-bit version of instruction
↳ quad word

movl → 32-bit version of instruction

→ %eax %rax (64)

movl $16, %rax

constant ← first 32-bits set zeroed out (ignore)

---

operations

- transfer data between memory + register
  - Load data from memory into a register (MOV)
  - Store register data into memory
- Perform arithmetic functions on registers or memory

  (add, sub, mul)

- transfer Control
  - unconditional jumps to/from procedures
  - Condition braches
  - indirect branches (JMP)

Moving data

movq Source, Destination
Src, Dest

immediate operation
↳ the use of constants
↳ denoted $

↙

$-533

movq $-533, %rax ← immediate regular move

movq $-533, (%rax) ← immediate memory move

replace value at this memory address

movq %rdi, %rax ← regular regular move

address movement

movq (%rdi), %rax ← memory regular move

Value at this memory address

movq %rax, (%rdx) ← regular memory move

Simple addressing

movq (%rcx), %rax

↓

applying a displacement

↳ movq 8(%rcx), %rax

↳ (%rcx)+8

value at an address

Example of Simple Addressing

```
void swap (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    movq    (%rdi), %rax
    movq    (%rsi), %rdx
    movq    %rdx, (%rdi)
    movq    %rax, (%rsi)
    ret
```

*xp = rdi

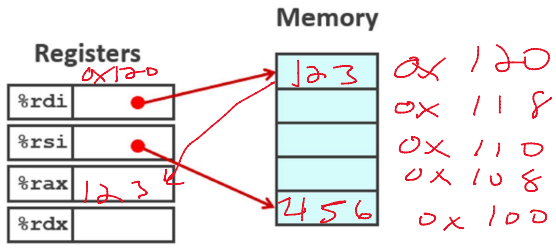
t0 = rax

*yp = rsi

t1 = rdx

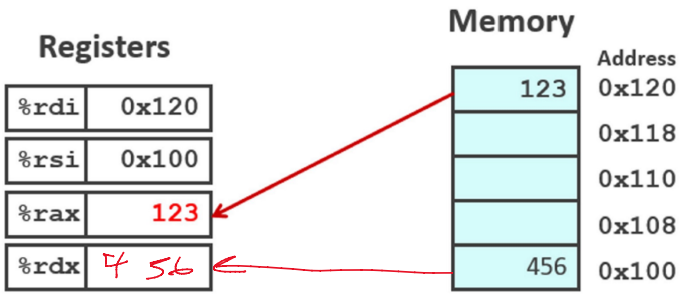
Understanding Swap()

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

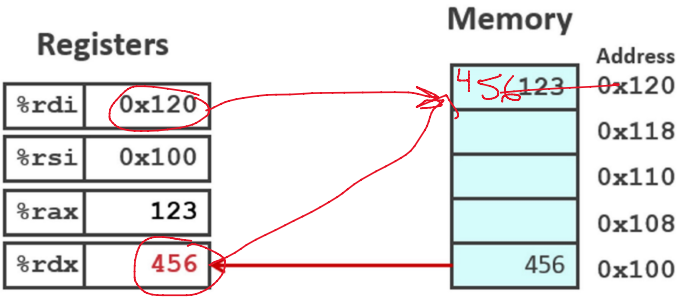


Register	Value
%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

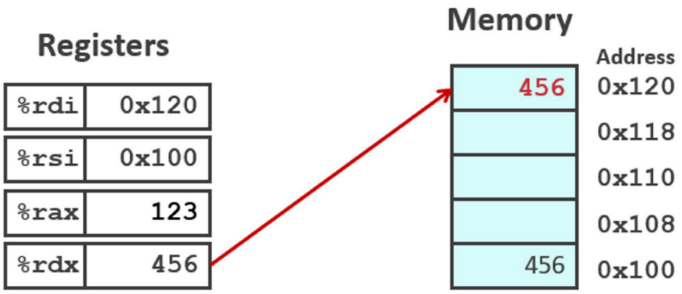



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

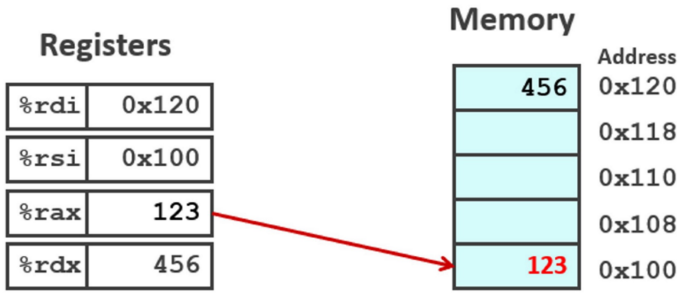



```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

Memory




```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```




```
swap:
   movq    (%rdi), %rax  # t0 = *xp
   movq    (%rsi), %rdx  # t1 = *yp
   movq    %rdx, (%rdi)  # *xp = t1
   movq    %rax, (%rsi)  # *yp = t0
   ret
```

General form of memory addressing

$Imm(Rb, Ri, S)$

Imm refers to a constant displacement

Rb → base register

Ri → index register

S → Scale 1, 2, 4, or 8

$imm(Rb, Ri, S) = Mem[Reg[Rb] + S*Reg[Ri] + Imm]$

Memory lookup

address computation

Address Computation Examples

%rdx	0xf000
%rcx	0x0100

Expression	Address Computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)	0xf000 + 0x0100	0xf100
(%rdx,%rcx,4)	0xf000 + 4*0x0100	0xf400
0x80(,%rdx,2)	0xf000*2 + 0x80	0x1e080

0xf000 + 0x8

256*4 = 1024

$16^2 * 4$

0xF000
+0xF000
0x1e000
+ 0x80
0x1e080

$16^3 * (F*2)$
↳ 1e000

Load effective address

lea src, dest

↳ you can do computation operations

leaq 8(%rdi, %rdi, 2), %rax

↑ return register

assume rdi = x in c

x + 2*x + 8

→ 3x + 8 → %rax

optimization

```
movq $5, rax
addq %rdi, %rax
ret
```

↓ in c

↓ in c

rax = a
rdi = b

a = 5

→ a = 5 + b

Arithmetic operations: in c

addq src, dest → Dest = Dest + Src

Subq src, dest → Dest = Dest − Src

imulq src, dest → Dest = Dest * Src

1 operand instructions

incq Dest → Dest = Dest + 1

decq Dest → Dest = Dest − 1

negq Dest → Dest = −Dest

notq Dest → Dest = ~Dest

Jump - change flow of program

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)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Cmp → Arithmetic operation → Set a flag

Test →

```
void op1(void);
void op2(void);
void decision(int x) {
        if (x) {
                op1();
        } else {
                op2();
        }
}
```

x = ! 1

L1

L2

```
op1:
  ⋮
op2:
  ⋮
decision:
  testq %rdi, %rdi
  je .L2
    call op1
    jmp .L1
.L2
```

Call op2
.L1
ret

X = rdi;
Y = rsi;

```
long absdiff
  (long x, long y)
{        rax
  long result;
  if (x > y) <=
    result = x-y;
  else L4
    result = y-x;
  return result;
}
```

absdiff:
Cmpq %rsi, %rdi
jle .L4
x → result ← Movq %rdi, %rax
result=x-y ← Subq %rsi, %rax
ret
.L4
Movq %rsi, %rax
Subq %rdi, %rax
ret

5. Consider the following assembly code:

```
# count = edi
loop:
    cmpl   $9, %edi
    jle    .L4
    movl   %edi, %eax
    ret
.L3:
    addl   $3, %edi
    cmpl   $9, %edi
    jg     .L7
.L4:
    cmpl   $8, %edi
    jle    .L3
    leal   1(%rdi), %eax
    ret
.L7:
    movl   %edi, %eax
    ret
```

Fill in the resulting code:

```
int loop(int count) {
    while (_____) {
        if (count _____)
            count _____;
        else
            count _____;
    }
    return count;

}
```

In C you have goto statement that allow you to jump to labels

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y) <=
    result = x-y;
  else
    result = y-x;
  return result;
```

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

```
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
long result;
if (x <= y)
    goto Else;
result = x - y;
goto Done;
Else:
    result = y - x;
Done:
    return result;
}
```

"Do-While" Loop Example

C Code

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

Goto Version

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

Exercise #1

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x104	0xAB	%rcx	0x1
0x108	0x13	%rdx	0x3
0x10C	0x11		

Fill in the following table showing the values for the indicated operands:

Operand	Value
%rax	0x100
0x104	0xAB
$0x108	0x108
4(%rax)	(0x100+4) = 0xAB
9(%rax, %rdx)	(0x100+0x3)+9 = 0x103+9 = 0x10C = 0x11
0xFC(%rcx,4)	(0x1*4)+0xFC = 0x4+0xFC = 0x100 → 0xFF
(%rax, %rdx, 4)	0x100 + 0x3x4 = 0x100+0xC = 0x10C = 0x11

Exercise #2

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x108	0xAB	%rcx	0x1
0x110	0x13	%rdx	0x3
0x118	0x11		

Fill in the following table showing the effects of the following functions (both register or memory location):

Instruction	Destination	Value
addq %rcx, (%rax)	0x100	0x100
subq %rdx, 8(%rax)	0x108	0xA8
imulq $16, (%rax, %rdx, 8)	0x118	0x110
incq 16(%rax)	0x110	0x14
decq %rcx	%rcx	0x0
subq %rdx, %rax	%rax	0xFD

addq %rcx, (%rax) = 0x1 + (0x100) = 0x1+0xFF = 0x100

subq %rdx, 8(%rax) $= (0x100 + 8) - 0x3 = 0xA3 - 0x3 = 0xA8$

imulq $16, (%rax, %rdx, 8) $= (0x100 + 0x3 * 8) * 16$

$= (0x100 + 0x18) * 16$

$= (0x118) * 16$

$= 0x11 * 16 = \boxed{0x110}$

incq 16(%rax) $= ((\%rax) + 16) + 1$

$16^{1} = 0x10$

$= (0x100 + 16) + 1$

$= (0x110) + 1$

$= 0x13 + 1$

$= \boxed{0x14}$

decq %rcx $= \%rcx - 1$

$= 0x1 - 1$

$= 0x0$

subq %rdx, %rax

0x3 0x100

$= 0x100 - 0x3$ ← subtract 1

$= 0xFF - 0x2$

$= \boxed{0xFD}$