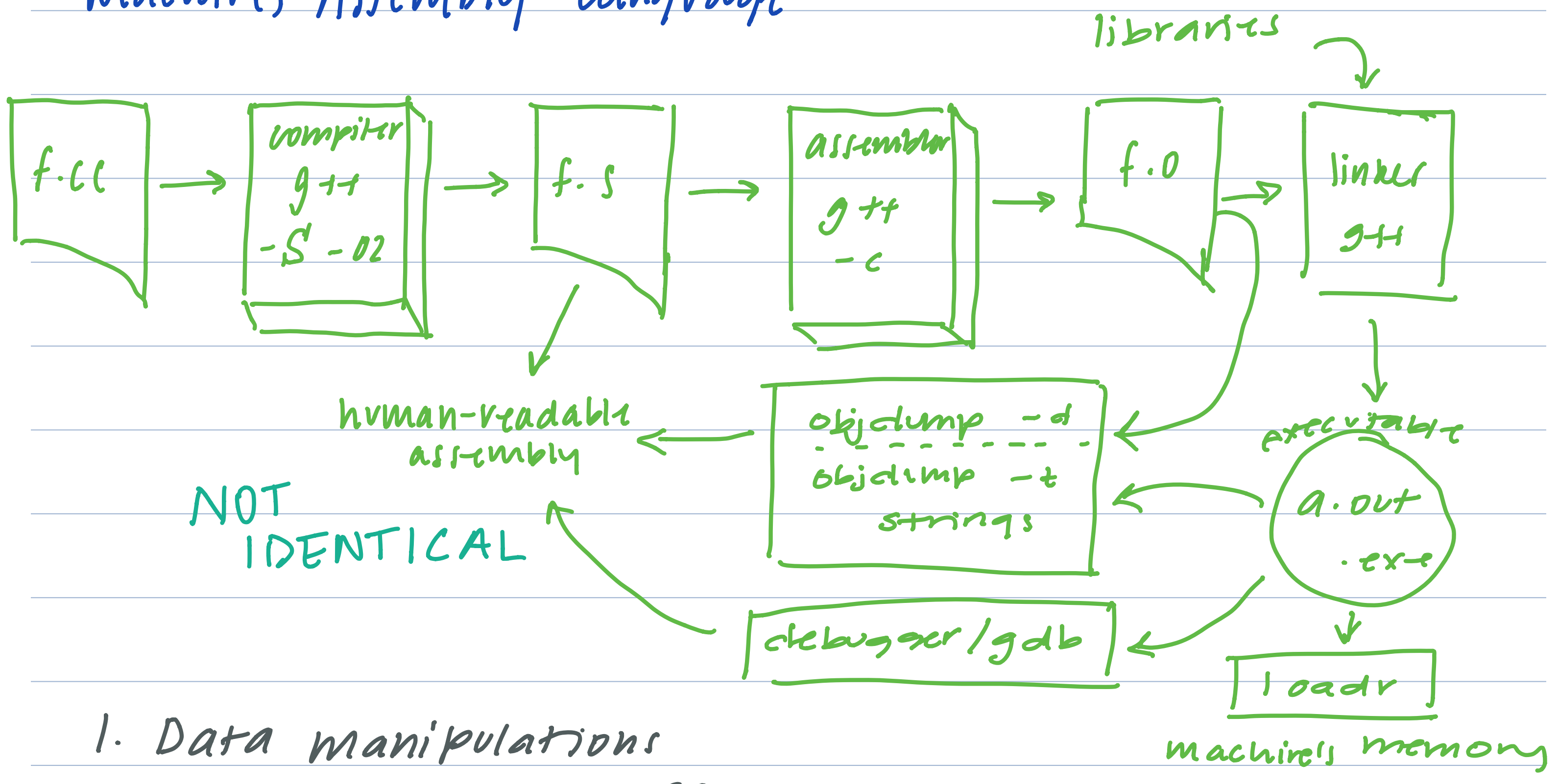
CS 61 Lecture 7

September 24, 2019

Machine, Assembly language



1. Data manipulations
2. Data movement (eg. assignment)
3. Control flow (eg. if statements)

Assembly code * . ___ is an assembler direct, not part of program

```
        .file    "foo.cc"
        .text          program is following
        .globl   _Z1fv
        .type    _Z1fv, @function
_Z1fv:   label
.LFB0:
        movl $100, %eax
        rep ret   return
.LFE0:
        .size    _Z1fv, .-_Z1fv
        .indent     ...
        .section    ...
```

xorl %eax, %eax → xor op1, op2

op1 ↓ source, op2 ↓ source/dest

op2 = op2 ^ op1

Structure

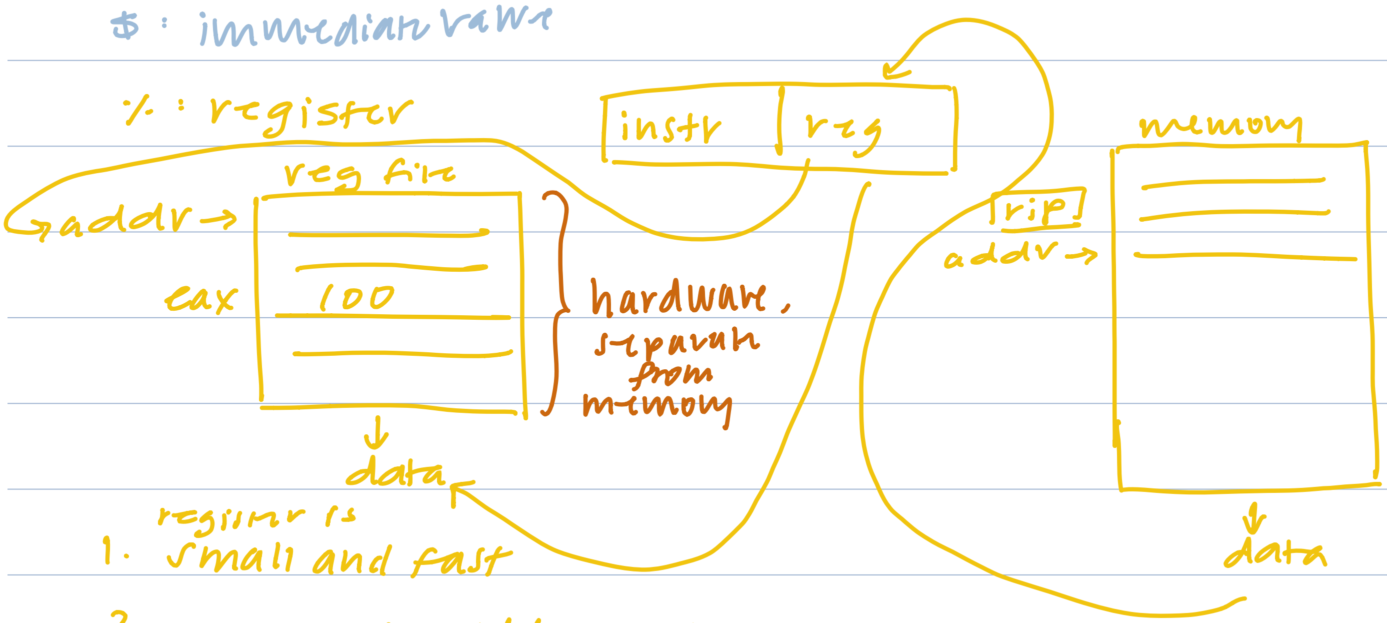
instruction = opcode operands

$$\begin{tikzcd} \text{operands} \arrow[d] \\ \text{src} \arrow[r] & \text{dst} \end{tikzcd}$$

movl: move

$: immediate value

% : register

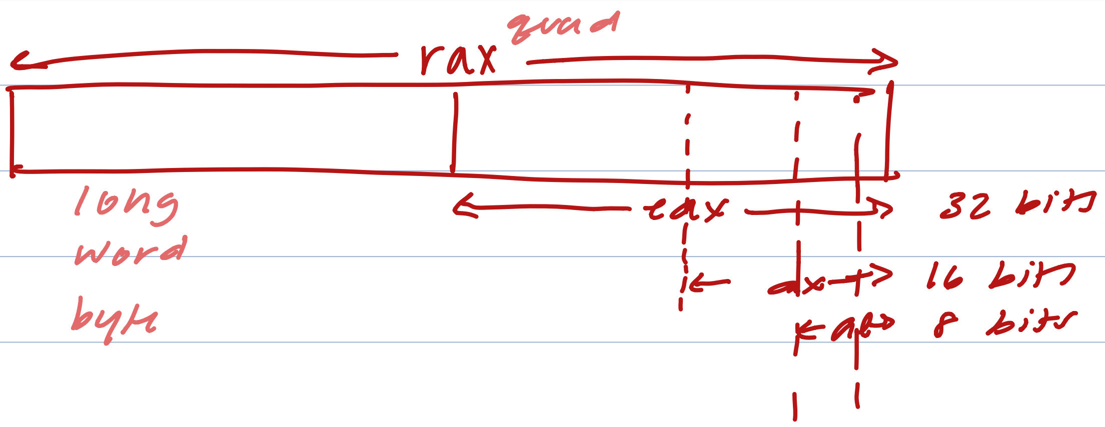


register is
1. small and fast

2. register is hidden under abstraction

```
g++ -c f01.s ; objdump -d f01.o
```

64- bit architecture



Software convention: eax sometimes the return value/ register.

```
movl  b(%rip), %eax
addl  a(%rip), %eax
ret
```

extern

instruction pointer

global variable

function to add a and b

32 bits

```
movl  aq (%rip), %eax
addl  ai (%rip), %eax
addl  au (%rip), %eax
addl  bi (%rip), %eax
ret
```

→ single instruction gives information about size but nothing else

move long (32 bits) to rax (64 bit register) and duplicate bit 31 for 32 bits to preserve sign

```
movslq  b(%rip), %rax
addq    a(%rip), %rax
ret
```

→ add quad (32 bits) to quad (32 bits)

POINTER ARITHMATIC