assembling language.

(I) History of Intel processors and architectures

(II) C, assembly, machine code

(III) Assembly Basics: Registers, operands, move

(IV) : Arithmetic & logical operations,

Intel x86 Processors

(I) Dominate laptop/desktop/server mkt

(II) Evolutionary design:
  (i) Backwards compatible up until 8086, introduced in 1978
  (ii) Added more features as time goes on.

(III) Complex instruction set computer (CISC)
  (i) Many different instructions with many different formats
  But, only small subset encountered with Linux programs.

  (ii) Hard to match performance of Reduced Instruction Set computers (RISC) fight 1980s.

  (iii) But, Intel has just done that.
  In terms of speed. Less so for low power.

1985 IA32 First 32-bit Intel processor.

{ frequency
  multi-processor on a single chip (multicore)
  ‖
  Thank machine (2008)

cache: a temporary made used to hold the most recently accessed data, so you can access it more quickly.

Definitions:

(I) Architecture (also ISA: instruction set architecture) The parts of a processor that one needs to understand or write assembly/machine code.

e.g. Instruction set specification, registers.

(II) Microarchitecture: implementation of the architecture.

e.g. cache sizes and core frequency.

(III) Code forms:

(i) machine code: the byte-level programs that a processor executes

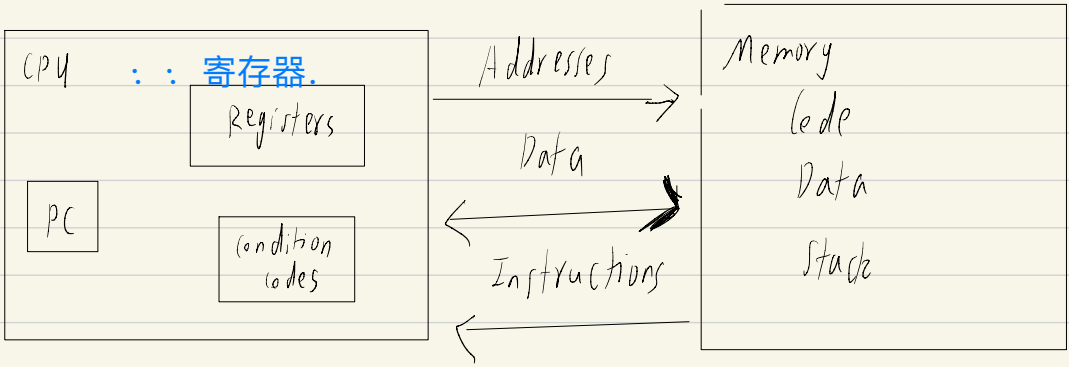
(ii) Assembly code: A text representation of machine code.

Example ISAs:

(i) Intel: X86, IA32, Itanium, X86-64.

(ii) ARM: Used in almost all cellphones.

Assembly / Machine Code View



Programmer-Visible State:

(I) PC: Program counter
- (i) Address of next instruction
- (ii) called "RIP" (x86-64)

(II) Register file:
- (i) Heavily used program data (give name to)

(III) Condition codes:

Some status information about most recent arithmetic or logical operation.
Data for conditional branching.

logically, an array of bytes.

(I) Memory:
- (i) Byte addressable array
- (ii) Code and user data
- (iii) Stack to support procedures.

virtual memory:
seems that each project independent memory, ...
but physically, share the same memory ...

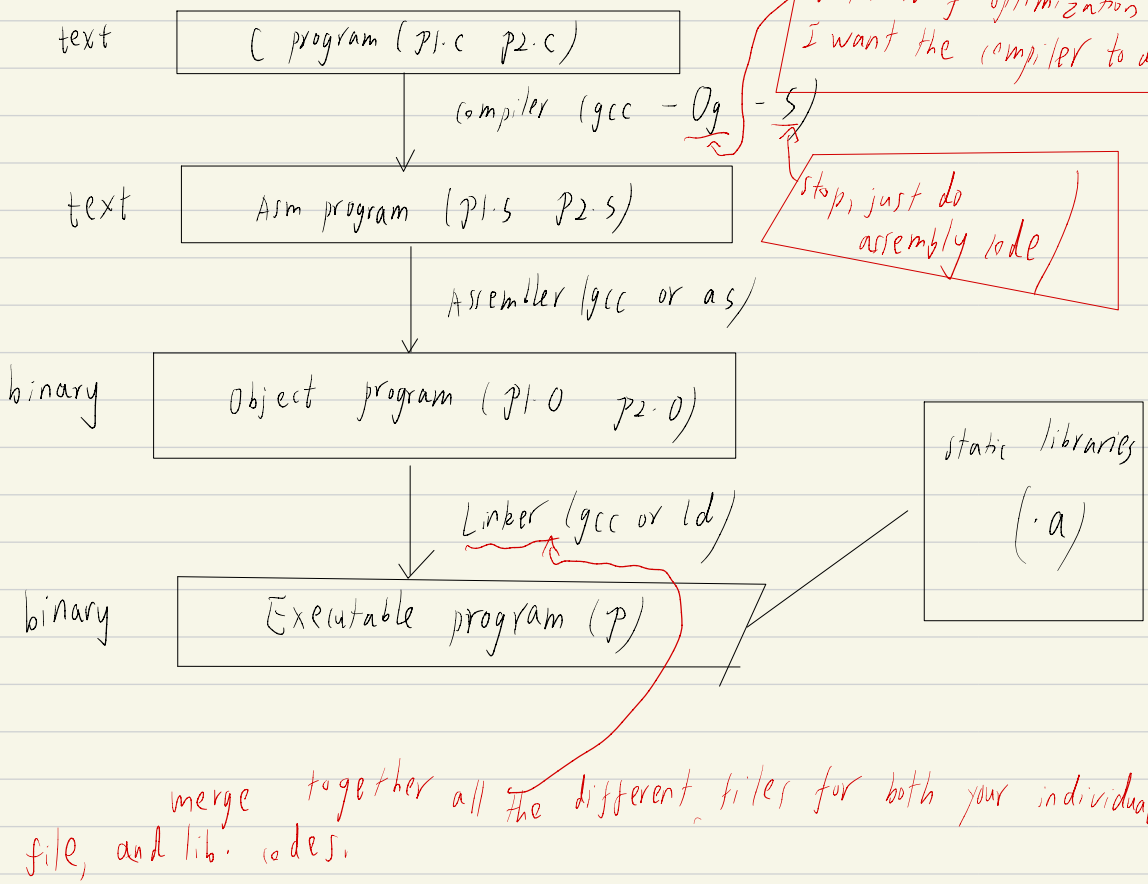
Turning C into Object Code.

(I) Code in files p1.c p2.c

(II) Compile with command: gcc -Og p1.c p2.c -o p

(i) Use basic optimization (-Og) [New to recent versions of GCC]

(ii) Put resulting binary in file p.



merge together all the different files for both your individual file, and lib. codes.

Assembly Characteristics: Data Types

(I) "Integer" data of 1, 2, 4 or 8 bytes
- (i) Data values
- (ii) Addresses (untyped pointers)

→ stored as Numbers in computer.

(II) Floating point data of 4, 8 or 10 bytes

(III) Code: Byte sequences encoding series of instructions

(IV): No aggregate types such as arrays or structures
- (i) just contiguously allocated bytes in memory

```
gcc    -S    sum.c
and
gcc    -c    sum.c
```

Assembly Characteristics: Operations

(I) Perform arithmetic function on register or memory data

(II) Transfer data between memory into register.
- (i) Load data from memory into register
- (ii) Store register data into memory

(III) Transfer control
- (i) Unconditional jumps to/from procedures
- (ii) Conditional branches. (e.g. do which if switch

Object Code:

Code for sumstore:

0x0400595:

0x55
0x48
0x89
0xd3
0xe3
0xf2
0xff
0xff
0x48
0x89
0x03
0x56
0xc3

(*) Total of 14 bytes.
(*) Each instruction: 1, 3 or 5 bytes.
(*) starts at address 0x0400595

[I] Assembler

(i) Translates .s into .o
(ii) Binary encoding of each instruction.
(iii) Nearly-complete image of executable code.
(iv) Missing linkages between code in different files

[II] Linker:

(i) Resolves references between files
(ii) combines with static run-time libraries
Eg. code for malloc, printf.

(iii) Some libraries are dynamically linked
(i) Linking occurs when program being. execution

Machine Instruction Example.

```
*dest = t;
```

(I) C Code:
(i) store value t where designated by dest.

```
movq %rax, (%rbx)
```

(II) Assembly:
(i) Move 8-byte value to memory
(ii) quad words in x86-64 parlance.
(iii) Operands
t: Register %rax
dest: Register %rbx
*dest: Memory m[%rbx]

```
0x40059e: 48 89 03
```

(III) Object code
3-byte instruction
stored at address 0x40059e.

Disassembling Object code

Disassembled:

(I) Disassembler:

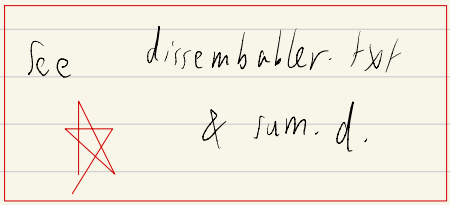
objdump -d sum

(i) Useful tool for examing object code

(ii) Analyzes bit pattern of series of instructions.

(iii) produces approximate rendition of assembly code

(iv). Can be run on either a.out (complete executable) or .o file.



What can be Disassembled?

(I) Anything that can be interpreted as executable code.

(II) Disassembler examines bytes and reconstructs assembly source

X86-64 Integer Registers

64 bits	32 bits	64 bits	
%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%edi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rdp	%edp	%r15	%r15d

stack pointers

can reference low-order 4 bytes (also low-order 1 & 2 bytes)

(I) Moving Data:

movq source, dest

(II) Operand Types.

(i) Immediate: constant integer data.

(ii) Example: $0x400 $-533.

(iii) Like C constant, but prefixed with '$'.

(ii) Register: One of 16 integer registers.

(i) Example: %rax, %r13

(ii) But %rsp reserved for special use.

(iii) Others have special uses for particular instructions.

(iii) Memory: 8 consecutive bytes of memory at address given by register.

(i) simplest example (%rax)

(ii) Various other "address modes".

%rax
%rcx
%rdx
%rsi
%rdi
%rsp
%rbp
%rN

movq Operand combinations:

movq source	dest	src, Dest	Analog
Imm	Reg	movq $0x4, %rax	temp = 0x4;
	Mem	movq $-147, (%rax)	*p = -147;
Reg	Reg	movq %rax %rdx	temp2 = temp1;
	Mem	movq %rax, (%rdx)	*p = temp;
Mem	Reg	movq (%rax), %rdx	temp = *p;

Cannot do memory transfer with a single instruction.

Simple Memory Addressing Modes.

(I) Normal (R) Mem[Reg[R]]

- Register R specifies memory address
- Aha! Pointer dereferencing in C.

```
movq (%rcx), %rax.
```

↑ dereferencing a ptr. and put it into a temporary.

(II) Displacement D(R) Mem[Reg[R] + D]

- Register R specifies start of memory region
- Constant displacement D specifies offset

```
movq 8(%rbp), %rdx
```

displacement D

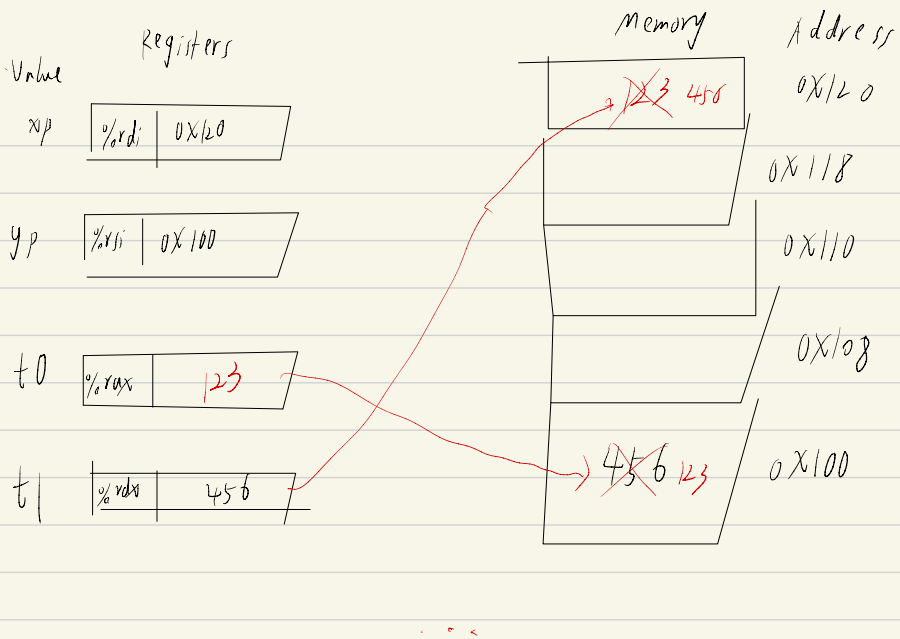
add 8 to get an address slightly off it by some fixed amount.

e.g.

```
void swap
    (long *xp, long *yp)
{
  long t0 = *xp //read memory into register
  long t1 = *yp
  *xp     = t1 // ...
  *yp     = t0 // copy back, but reverse ...
}
```

⇒

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


Registers
Value
xp
%rdi
0x120
yp
%rsi
0x100
t0
%rax
123
t1
%rdx
456
memory
Address
123 456
0x120
0x118
0x110
0x108
456 123
0x100

Complete Memory Addressing Modes

(I) Most General Form:

D(Rb, Ri, S) Mem[Reg[Rb] + S * Reg[Ri] + D]

(i) D: Constant "displacement" 1, 2, or 4 bytes

(ii) Rb: Base register: Any of 16 integer registers

(iii) Ri: Index register: Any, except for %rsp

(iv) S: Scale: 1, 2, 4 or 8 (why these numbers?)

A natural way to implement array.

int type: 4

long type: 8.

(II) Special cases:

(Rb, Ri) Mem[Reg[Rb] + Reg[Ri]]

D(Rb, Ri) Mem[Reg[Rb] + Reg[Ri] + D]

(Rb, Ri, S) Mem[Reg[Rb] + S * Reg[Ri]]

Address computation Examples.

%rdx 0xf000

%rcx 0x0100

Expression	Address computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx, %rcx)	0xf000 + 0x100	0xf100
(%rdx, %rcx, 4)	0xf000 + 4 * 0x100	0xf400
0x80(, %rdx, 2)	2 * 0xf000 + 0x80	0x1e080

Address Computation Instruction

(i) leaq Src, Dst

Src is address mode expression.
Set Dst to address denoted by expression.

(ii) Use

(i) Computing address without a memory reference
E.g., translation of p = &x[i]

(ii) Computing arithmetic expressions of the form x + k*y

k = 1, 2, 4 or 8.

Example

```
long m12 (long x)
{
    return x*12;
}
```

⇒

```
leaq (%rdi, %rdi, 2), %rax #
salq $2, %rax
```

1% rax := %rdi + 2*%rdi

shift left by 2

mov: { could either (1) refer to a register...
(2) or a memory location.

See some_arithmetic_operations.txt

The code you generated will correctly receive your implementations but it might not exactly replicate at a low level the exact sequence of the operations you specified at a high level.

Glossary:

Quadword: A numerical value of four times the magnitude of a word, thus typically 64 bits.