# Assembly Language

## 王爽

p3. 指令和数据放在内存中，CPU控制整个计算机的运作并进行运算

p5,CPU和内存间有地址总线，数据总线，控制总线。其中地址总线决定CPU的寻址能力，也就是支持多达的内存。数据总线决定CPU和外界传输数据的速度，8086有16跟数据总线，一次可以传送16位数据。控制总线决定CPU对外部器件的控制能力。

P9，内存分为随机存储器和只读存储器，前者可以读写，后者只能读，向ROM写入数据是没有作用的，ROM数据不会被改变。

P10，CPU对外部器件发出命令是通过控制总线

# Python

## Python核心编程

P34, 当不确定python在哪个目录下时，可以使用/bin/env python

P41, python的标准库文件安装在/usr/lib/python2.7目录

P43, 计算绝对值abs(-43)

P43, 定义字符串myString = ‘This is a test String’

P44, \_表示最后一个表达式的值

P44, 打印输出的格式有:

print ‘%s version is %d’%(“Python”,2)

print ‘your stirng is:’, user

Btw, 传递错误的类型并不会导致严重错误

P44, >>用于重定向

P44, 标准输入user=raw\_input(‘Enter login name: ‘)

P45, #表示comment

P48, [n:m]表示[n,m)区间

P48, +代表拼接字符串，\*代表重复字符串，比如’-‘\*20表示-----------------------

P49, aList=[1,2,3,4]是列表，aTuple=(1,2,”a”,4)表示元组。列表元素可以更改,元组不可以

P50, aDict={‘host’:’earth’}表示字典, aDict.keys()表示键值表。遍历字典：

For key in aDict:

print key, aDict[key]

p50,

if x < .0:

print “xxx”

elif x ==0:

print “xxx”

else:

print “xxx”

counter = 0:

while counter<3:

counter += 1

for item in [‘email’, ‘net’, ‘homework’]:

print item

for eachNum in [0,1,2]:

print eachNum

for eachNum in range(3):

print eachNum

for I, ch in enumerate(“abc”):

print ch, ‘%d’%i

p53, 列表解析：square=[x\*\*2 for x in range(6)]

P54, 读文件时open函数会把换行符一起读入，因此用print输出时需要加,来抑制重复换行

handle=open(file\_name, access\_mode=’r’)

如果不提供mode，默认是r，后续的文件操作都必须通过open返回的句柄

for eachline in handle:

print eachline,

handle.close()

P57，所有名字开始和结束都有两个下划线的函数都是特殊方法

类实例被创建时被执行，但不是构建器，而是用来初始化的.可以给用户自定义用来覆盖什么都不做的默认实现

\_\_init\_\_()

p57, self是实例自身的引用

p57, 创建实例fool = FooClass()

P58, import模块时，这个模块应该被放置在我们输入它的程序的同一个目录中，或者在sys.path所列目录之一。

P65, Python的赋值并不会返回值

P66, (x, y, z)=(1, 3, ‘a string’)

Q1: Python中单引号，双引号，三引号的区别

# Continous Integration

## Pro Git

### 常用命令

#获得远程分支

git remote show (remote)

#从远程抓取分支

git fetch origin

#将修改提交到远程分支

git push origin serverfix

git push origin serverfix:awesomebranch

#将远程分支合并到本地分支

git merge origin/serverfix

#如果合并的时候建立新的分支

git checkout -b serverfix origin/serverfix

git checkout --track origin/serverfix

git branch -u origin/serverfix

#查看跟踪分支

git branch –vv

#删除分支

git push origin --delete serverfix

### 初始化depot

git init

git config --global user.email "13240943@qq.com"

git config --global user.name "JM Han"

git add note/\*

git commit -m 'add note folder'

git remote add ebook https://github.com/dracohan/ebook.git

git remote set-url --add ebook https://github.com/dracohan/ebook.git

git remote –v

git push --set-upstream ebook master

git push ebook

## Practical Perforce

### 通用规则

1.1.2 …表示所有当前一级以及下一级文件

1.2 p4的命令参数，如果用双引号修饰，表明在传给p4之前不进行展开

### 常用命令

#查看用户信息

vmw\_ldap\_info hanb

#登录用户

p4\_login -Aa

#设置workspace

p4 client

#查看p4server 信息

p4 info

#设置p4 client

p4 set P4CONFIG=p4.config

Next, create files named p4.config in each of your workspace root directories. In each file, set the value of P4CLIENT

#查看文件

p4 files "//..."

#查找文件(包括delete的)

p4 files "//.../if.py"

#列出所有本地存在的文件

p4 have //…

#查找文件（不包括delete的）

p4 have //.../if.py

#查看目录

p4 dirs "//\*"

#查看file的map关系

#查看filelog

p4 filelog "//.../if.py"

#查看filelog，并包含integration历史

p4 filelog -i "//.../if.py"

#查看文件内容随版本的变化, -c代表输出changelist，而不是revision

p4 annotate -c //public/revml/revml.dtd

#（只显示plain text）

p4 print //public/revml/revml.dtd

#查看更改过的文件

p4 diff -sa

#查看integrate和resolve的文件

p4 diff -sb

#查找丢失的文件

p4 diff -sd

#强制sync未被打开但是丢失的文件, -x-使p4读取标准输入，而不是p4的参数

p4 diff -sd | p4 -x- sync -f

#check out未被打开但是更改过的文件

p4 diff -se | p4 -x- edit

#查看所有未被打开的文件的状态

p4 diff -sl

#查看打开了但是与depot相同的文件

p4 diff -sr

#ignore空格键的不同

p4 diff -db

#ignore空格键的存在与否

p4 diff -dw

#比较depot的两个文件

p4 diff2

p4 diff2 //public/revml/README#2 //public/revml/README#3

#查看p4文件类型

p4 help filetypes

#编辑

p4 edit index.html locations.html

#查看已经打开的文件

p4 opened

#查看谁在更改文件

p4 opened -a if.py

#查看更改的内容

p4 diff

#互换两个文件的内容

p4 integrate -i readme.txt misc.txt

p4 integrate -i misc.txt readme.txt

p4 resolve -at

p4 submit

#打印文件的内容, -q是省略p4加的文件头

p4 print -q ads.html#5

#增加文件

p4 add using\_mod.py

p4 add \*.py

p4 submit

#增加整个目录的文件

dir /s/b/a-d python | p4 -x- add -f

#删除

p4 delete using\_func.py

p4 submit

#提交修改

p4 submit

p4 submit -c 1406

p4 submit python/... #提交某一个目录下的所有修改, 必须在default changelist中执行

#新建changelist

p4 change

#删除空白changelist

p4 change -d 1406

#查看pending的changelist

p4 changes -s pending -c scriptws

#移动文件到changelist中

p4 reopen -c 28 //.../\*

#查看changelist中的修改, -c表示希望的changelist

p4 opened -c 28

#查看文件的changelist

p4 changes //depot/.../\*.html

#查看最近的5个修改

p4 changes -m5 "//.../if.py"

#查看changelist的信息

p4 describe -s 1

#查看changelist影响的文件

p4 files @=3947829

# resolve冲突

p4 resolve

#查看需要resolve的file, -o表示显示base

p4 resolve -n -o

#自动resolve

p4 resolve -as

#只接受theirs的resolve

p4 resolve -at

#只接受你自己的版本, always accept yours

p4 resolve -ay

#让p4merge

p4 resolve -am

#查看resolved结果

p4 resolved -o

#revert

p4 revert

#revert所有没有修改的或者丢失的文件，注意，resolve过的不会被自动revert

#-a表示opened的文件，而不是revert后面的参数

p4 revert -a

#sync代码

p4 sync -n #预览要sync的内容

p4 sync -f #强制sync

p4 sync "//depot/.../\*.py"

p4 sync //depot/dev/www/...@2004/08/10

p4 sync [//.../if.py@26](mailto://.../if.py@26)

#在local删除文件，并让depot知晓，相当于git的--cached

p4 sync c:\ws\dev\www\...#none

#直接delete文件

p4 delete //depot/…/\*.html

#查看最近一个提交

p4 changes -m1

#查看大于某一个changelist的所有changelist

p4 changes ... "@>83478"

#查看本地文件sync到哪个版本

p4 changes -m1 ...#have

#branch

p4 integ //Ace/MAIN/...@2004/10/12 //Ace/V1/...

#revert branch

p4 revert //depot/script/v1/...

#只integrate某一个changelist

p4 integ //Ace/V1/...@3459,@3459 //Ace/MAIN/...

### #Revert一个修改(1245)

p4 sync @1244

p4 files @=1245 | sed -n -e "s/#.\* - delete .\*//p" | p4 -x- add

p4 files @=1245 | sed -n -e "s/#.\* - edit .\*//p" | p4 -x- edit

p4 sync

p4 files @=1245 | sed -n -e "s/#.\* - add .\*//p" | p4 -x- delete

p4 resolve -ay

p4 submit