Control Flow Instructions:

Branching/Looping purpose (Intra-procedure control flow)

```
   1) Unconditional Jump Instructions:
      jmp
   2) Conditional Jump Instructions
      i) Signed
         jl, jle, jg, jge, je, jne, jnl, jnle, jng, jnge
      ii) Unsigned
         jb, jbe, ja, jae, jz, jnz, jnb, jnbe, jna, jnae
      iii) Flag based
         jc, jnc, jz, jnz, js, jns, jo, jno
```

Inter-procedure (amongst different procedure)

```
   call
   ret
   ret $n
```

Inter-privilege level (app->kernel, kernel->app)

```
   x86: int  <interruptnumber>

   x64:   sysenter
          sysret
```

#-----------------------

Syntax and semnatics of jump instrution:

```
jxxx   Label
```

IMP RECALL: While executing any control flow instruction, microprocessor relies on that instruction to set the instruction pointer as a part of its execution. It does not use its default logic(that of addr of current instruction + size of current instruction) to compute the address of next instruction.

#--

Unconditional Jump Instruction:

```
jmp    adderess

jmp    label_in_text_section # Direct addressing mode

jmp    *%reg32
```

Let reg32 be one of the general purpose registers. It can be loaded with the base address of an instruction by following:

```
movl   $label_in_text_section, %reg32
jmp    *%reg32 # Indirect addressing mode

jmp    addr
```

Behaviour: As a part of hardware execution, jmp addr instruction sets eip to addr (eip <- addr). The jmp being a control flow instruction, the next instruction will be fetched from addr (address to which eip was set by the jmp instruction)

#--

Late binding:

```
Base* pB = new Derived;

pB->f();  // f() is virtual in base class and is overridden in derived class
          // pB->f(), D::f()
          // DLL

# Algorithm
# pB->f() which address
# movl    addr, %reg32
# call    *%reg32
#----------------------------
```

Conditional Jump Instruction

Signed

```
jl addr
jle addr
jg addr
jge addr
je addr
jne addr
```

addr can be a lable in text section (direct addressing mode)
addr can be *%reg (register indirect addressing mode)

Unsigned

```
jb addr
jbe addr
ja addr
jae addr
jz addr
jnz addr
```

```
jxxx ADDR
```

eip is set to ADDR if certain condition is satisfied (certain means which??)
eip is set to the address of next instruction (addr of current instruction + size of current instruction) if the condition is NOT SATISIFIED.

```
cmpl   src, dest
jxxx   addr
next instruction
#------------------

cmpl   src, dest
jl     L1
next instruction

L1:
   some block of instructions
```

```
Meaning:
   In PREVIOUS cmp instruction, if
   SIGNED(dest) < SIGNED(src)
   had been true THEN
   take a jump.
   Otherwise goto next instruction
#-----------------------------------------------
```

```
cmpl   src, dest
jle    L1
next instruction


L1:
   some block of instruction
```

Meaning of jle L1
In previous cmp instruction if
   SIGNED(dest) <= SIGNED(src) then set eip to $L1. Otherwise set eip to address

```
of next instruction.

#---------------------------

cmpl   src, dest
jg     L1
next instruction


L1:
   some block of instruction

Meaning of jg    L1

In previous cmp instruction if
   SIGNED(dest) > SIGNED(src)
then set eip to $L1 otherwise set eip to the address of the next instruction

#----------------------

cmpl src, dest
jge    L1
next instruction

L1:
   some block of instructions

Meaning of jge   L1
In previous cmp instruction if
   SIGNED(dest) >= SIGNED(src)
then set eip to $L1 (i.e. transfer control flow to L1)
otherwise set eip to the address of the next instruction.

#-----------

cmpl src, dest
je L1
next instruction

L1:
   some block of instructions

Meaning of
je L1
```

```
In previous cmp instruction of SIGNED(dest) == SIGNED(src) then
set eip to $L1 else set eip to the address of the next instruction.

#-------------

cmple src, dest
jne    L1
next instruction

L1:
   some block of instruction

Meaning of
jne    L1

In previous cmp instruction if SIGNED(dest) != SIGNED(src)
then set eip to $L1 else set eip to the address of the next instruction

#---------------------------------------------------

cmpl   src, dest
jb     L1
next instruction

L1:
   some block of instructions

Meaning:
   In PREVIOUS cmp instruction, if
   UNSIGNED(dest) < UNSIGNED(src)
   had been true THEN
   take a jump.
   Otherwise goto next instruction
#------------------------------------------------

cmpl   src, dest
jbe    L1
next instruction


L1:
   some block of instruction
```

Meaning of jbe L1
In previous cmp instruction if
UNSIGNED(dest) <= UNSIGNED(src) then set eip to $L1. Otherwise set eip to address
of next instruction.

```
#--------------------------

cmpl   src, dest
ja     L1
next instruction


L1:
   some block of instruction
```

Meaning of ja L1

In previous cmp instruction if
UNSIGNED(dest) > UNSIGNED(src)
then set eip to $L1 otherwise set eip to the address of the next instruction

```
#----------------------

cmpl src, dest
jae    L1
next instruction

L1:
   some block of instructions
```

Meaning of jae   L1
In previous cmp instruction if
UNSIGNED(dest) >= UNSIGNED(src)
then set eip to $L1 (i.e. transfer control flow to L1)
otherwise set eip to the address of the next instruction.

```
#-----------

cmpl src, dest
jz L1
next instruction

L1:
    some block of instructions
```

Meaning of

```
jz  L1
```

In previous cmp instruction of UNSIGNED(dest) == UNSIGNED(src) then set eip to $L1 else set eip to the address of the next instruction.

```
#-------------

cmpl src, dest
jnz    L1
next instruction

L1:
    some block of instruction
```

Meaning of

```
jnz    L1
```

In previous cmp instruction if UNSIGNED(dest) != UNSIGNED(src) then set eip to $L1 else set eip to the address of the next instruction

```
#-----------------------------

jl  == jnge
jle == jng
jg == jnle
jge == jnl

jb == jnae
jbe == jna
ja == jnbe
jae == jnb
```

2|5|0

c|2

Math!

Unix:

John v6

- ?PDP assembly
- Computer Zaks 1st edition

Intel

2
S
C
O

Rules

PDP.

- Mit opencourseware

PDP-7

PDP-11/70.

Manual PDF

DEC

An