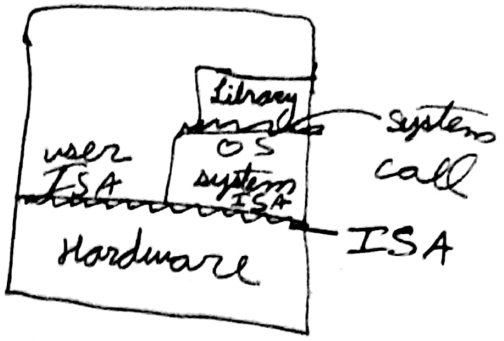
Theory of Virtualisation

James E Smith
Ravi Nagar

Abstraction

application



Drawback – application build for an OS cannot run on other OS.

Here comes virtualisation
a software required to map the instruction of windows corresponding to linux. It is called hypervisor, virtual software.

like JVM in java for portability.

x86, ISA
system call

hardware & software etc everything designed by different people yet work together

hardware provide ISA. instruction set architecture to user & the system. it is the interface b/w hardware & software.

App. Binary interface (ABI)

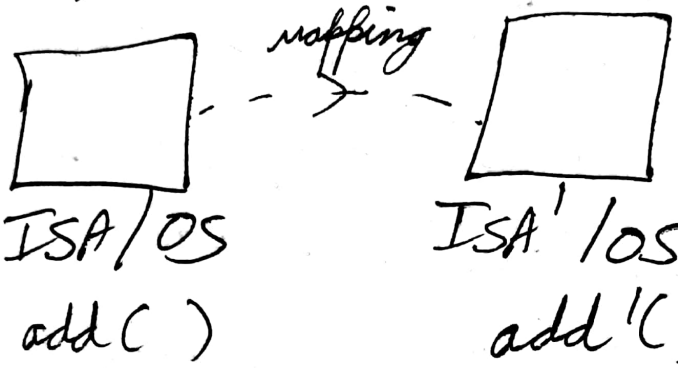
user application can only use user ISA.

to run applications on different systems well define ISA is required

complexity on implementation level is easy eg -:
windows MAC, x86 etc

Drawback -: application / software / process of one system does not work on other system.

How to run CSGO of windows on linux?
mapping the instructions



we have to transform add to add'( )

2 types of virtual machine -;
process virtual machine (support a process)
system virtual machine -> supports full ISA

JVM was made for portability.
byte code generated in java is portable i.e any machine having JVM can run this byte code.

CLI (common language infrastructure) in Dotnet as JVM in java.

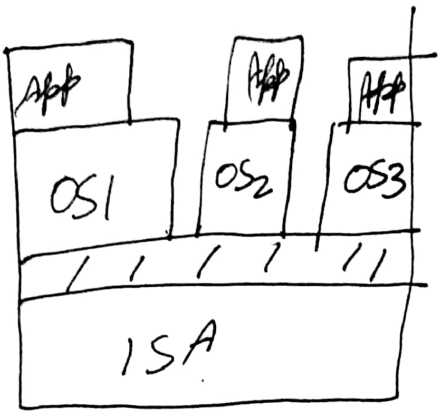
System virtual machine (support OS or ISA)

① classical system VM } same ISA

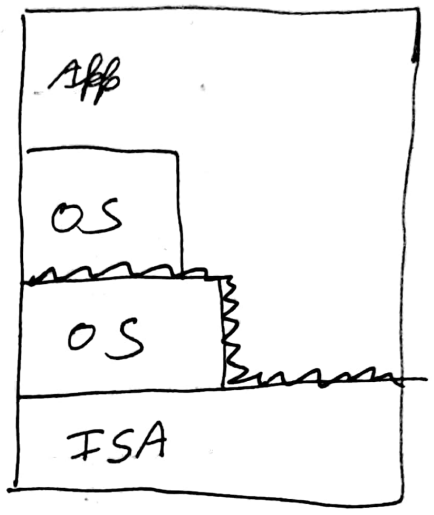
② Hosted VM (VMM)

③ whole system VM } different ISA

④ codesigned VM

① 

② eg- ~~Xen~~ Xen hypervisor



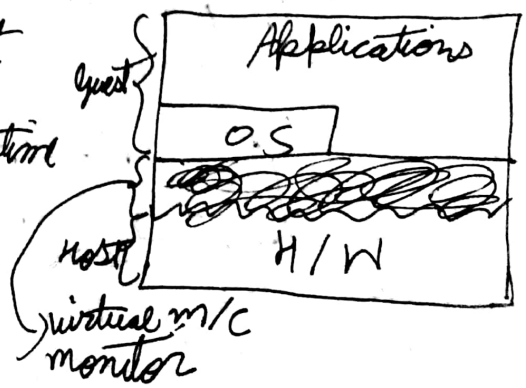
VM

Process VM — System VM

Process VM
- Emulation < Interpretation, Binary translation
- optimization: dynamic Binary optimizer
- High level language VM (JVM)

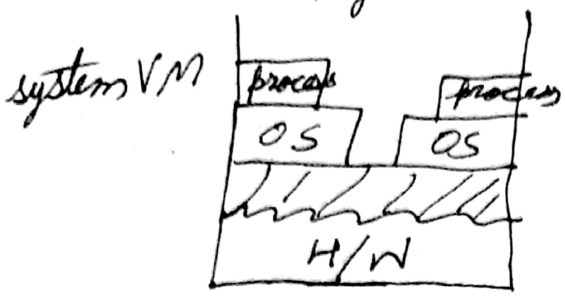
Process VM

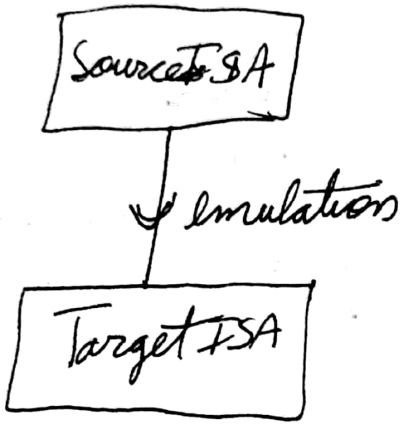
System VM

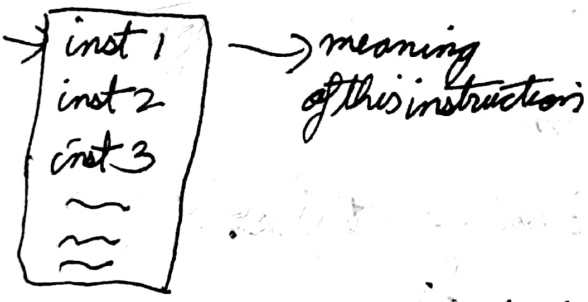


Same ISA — Multiprogramming
Same ISA Binary optimizer

Different ISA — Emulators
High Level language VM (SVM)



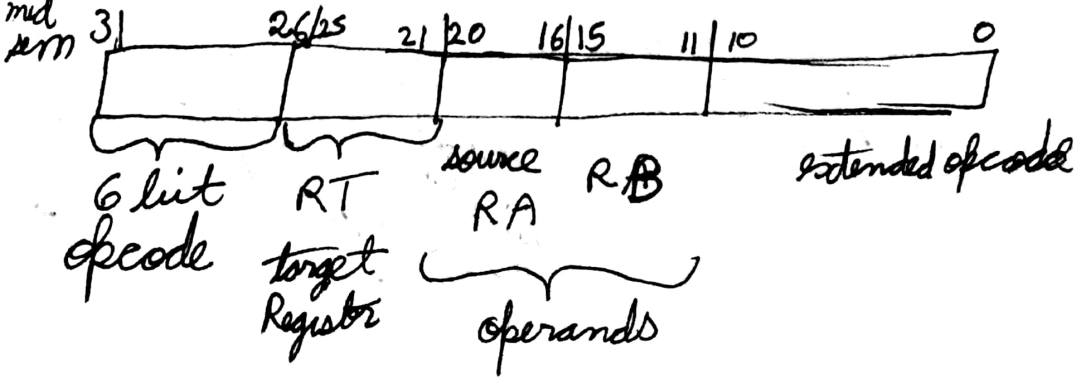




goal → to optimize interpretation

① Basic Interpretation

32 bit instructions



Branch Instruction
i/p-else or switch.

using branch instruction we used to interpret the code.

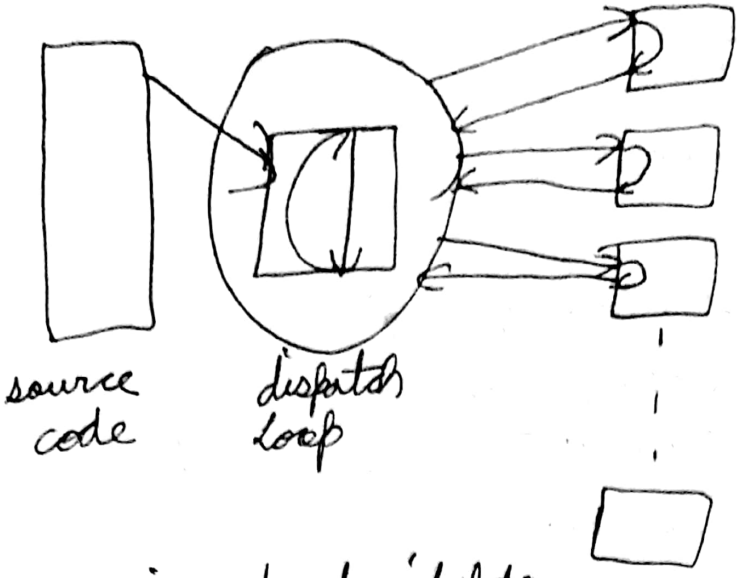
```
while(! halt && ! interrupt) {
    inst = code[PC];
    opcode = extract(inst, 31, 6);
    switch (opcode) {
        case Loadword & Zero:
        case ALU : ALU (inst);
        case Branch : Branch (inst);
        ....
    }
}
```

ppt - 11 block code

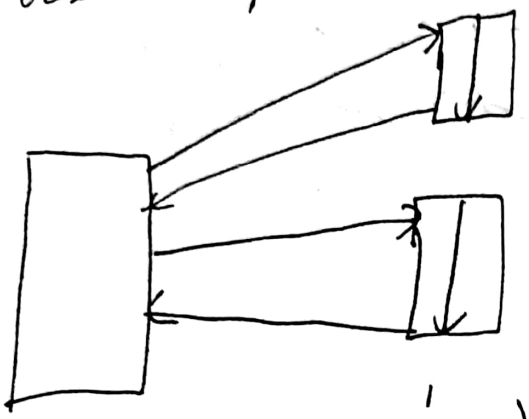
imitation,

our goal is to optimize the code interpretation

ALU(inst) is a function call & switch is a branch statement. These all are the overhead for the code.



version-1 decode-dispatch or basic interpretation — Interpret routine



version 2 threaded Implementation

disadvantage size of code is very large

Basic predecoding

interpretation means line by line whereas in binary translation whole code is converted altogether.

in threaded before goto there is a dispatch instruction

predecoding

opcode	RT	RA	RB

Direct thread Interpretation

PC routine	RT	RA	RB

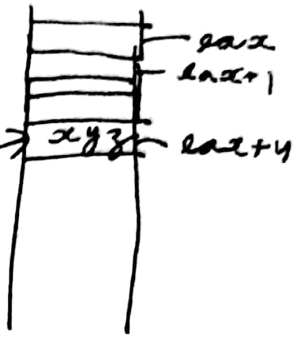
If opcode is add type ⇒ then on seeing the opcode function we are send to routine of that function

Limitation (direct thread)
① size of opcode is replaced by PC
∴ requires more space.
② If address of routine is replaced then we cannot go there.
③ portability is an issue in direct threaded

addl %edx, 4(%eax)

[eax+4]

xyz + edx

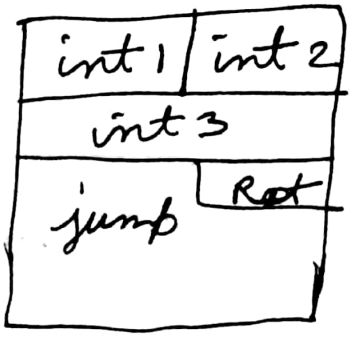


movl 4(%eax), %edx

[eax+4] ← edx

add $4, %eax

Code Discovery Problem :- from where the code start
code Location Problem :- where to jump



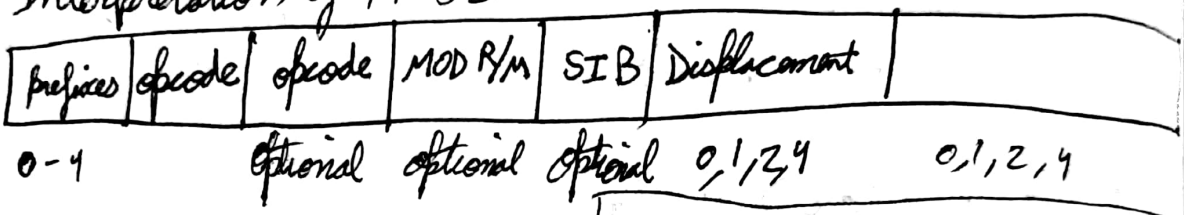
```
add
load
store
loop: load
      add
      store
      brcond skip
      load
      sub
```

Dynamic basic block is translated in a single go

ppt-30 (Lecture-2) Dynamic translation diagrams

SPC (source program counter)

Interpretation of 14-32



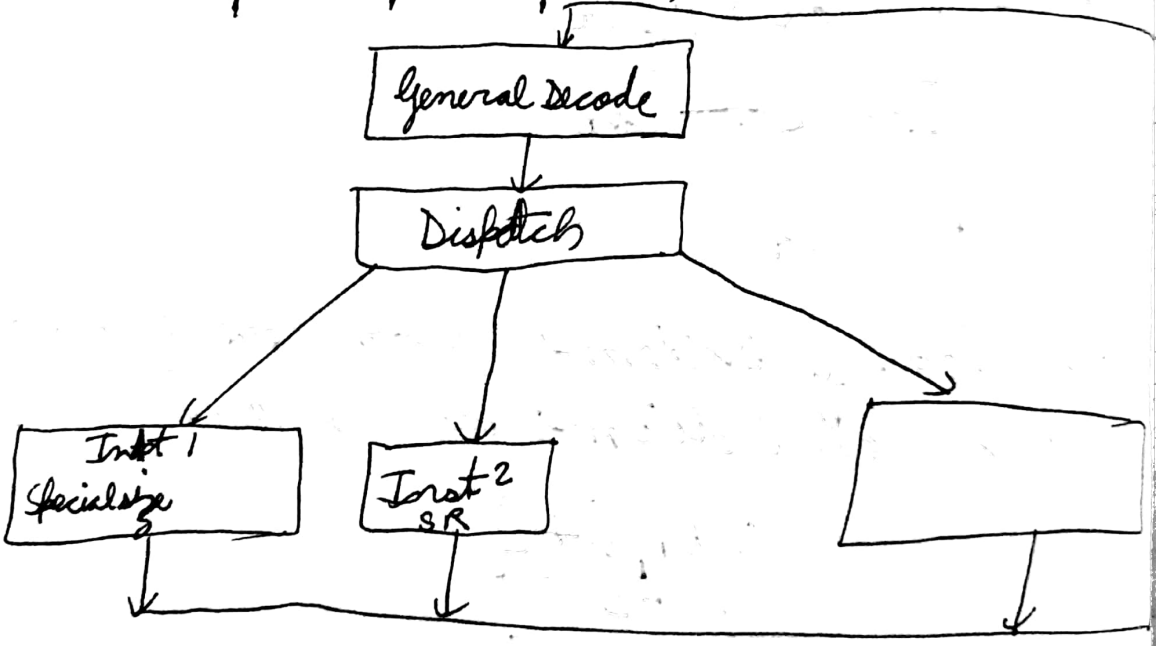
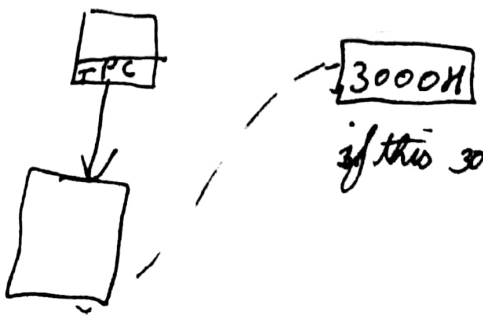


fig-38 to 48, 58-6 Disadvantage of S/W indirect jump prediction

fig-60, 61

2.7.2, 2.7.3



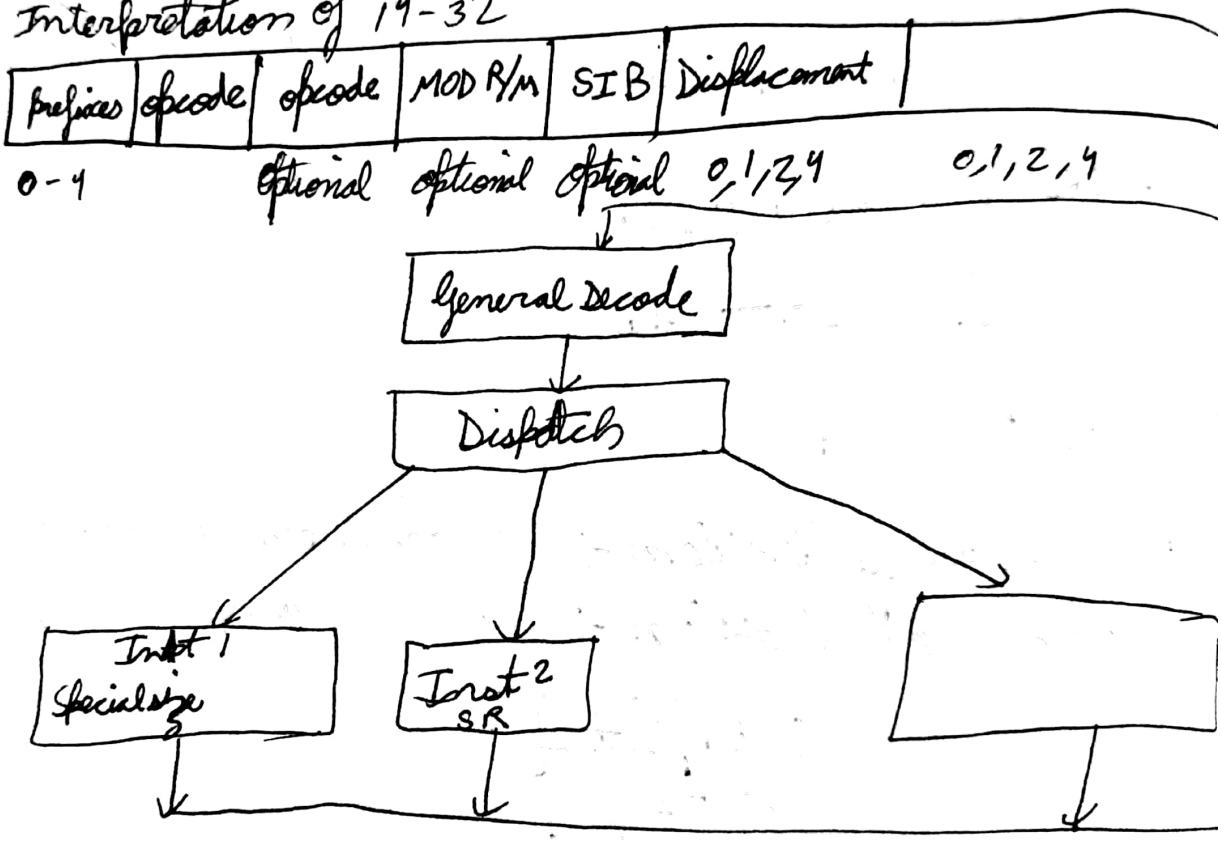
```
add
load
store
loop: load
      add
      store
      brcond skip
load
sub
```

Dynamic basic block is translated in a single go

ppt-30 (Lecture-2) Dynamic translation diagrams

SPC (source program counter)

Interpretation of 14-32



pg-38 to 48, 58-61 disadvantage of S/W indirect jump prediction

pg-60, 61

2.7.2, 2.7.3

we want mapping from SPC to TPC

where there is jump or branch statement, there starts a dynamic block

in register indirect jump, translation chaining will fail,

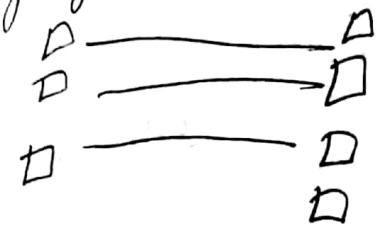
Instruction set issues

12 Feb '20

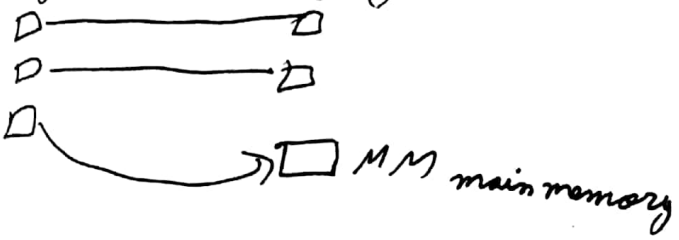
source ISA — target ISA

SISA — TISA

no of reg < no of reg easy



no of reg > no of reg



register is diff from main memory (time is more)

condition codes → tells state of accumulator

flag

SISA — TISA

Byte addressable — word addressable

network bigendian

little endian

29 Case Study ×