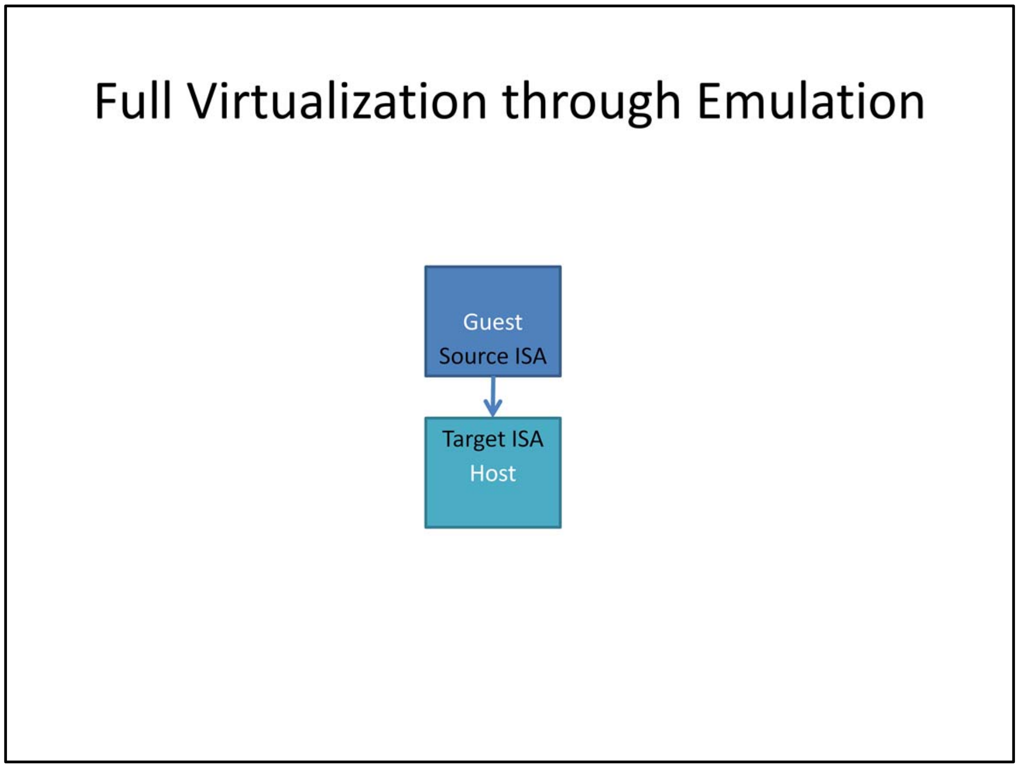
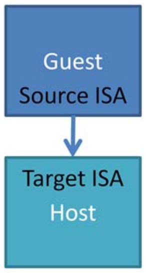
Interpretation Techniques in Virtualization

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

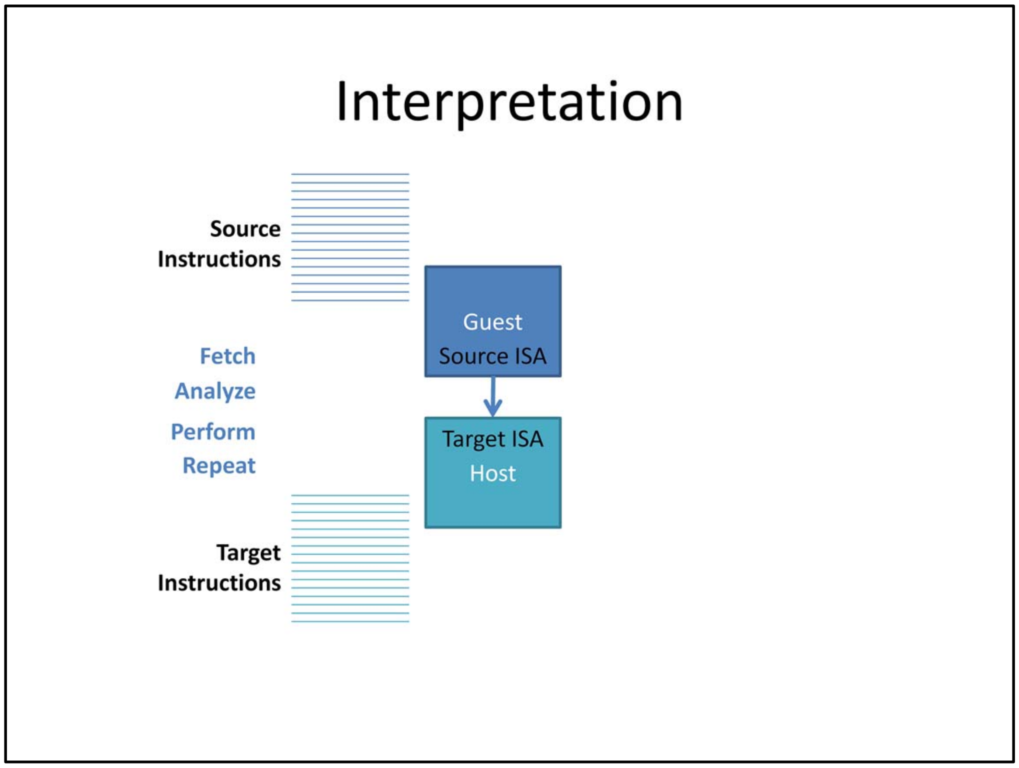
In this video, we shall look at some of the techniques used in interpretation of instructions from a guest to a host in virtualization.



Full Virtualization through Emulation

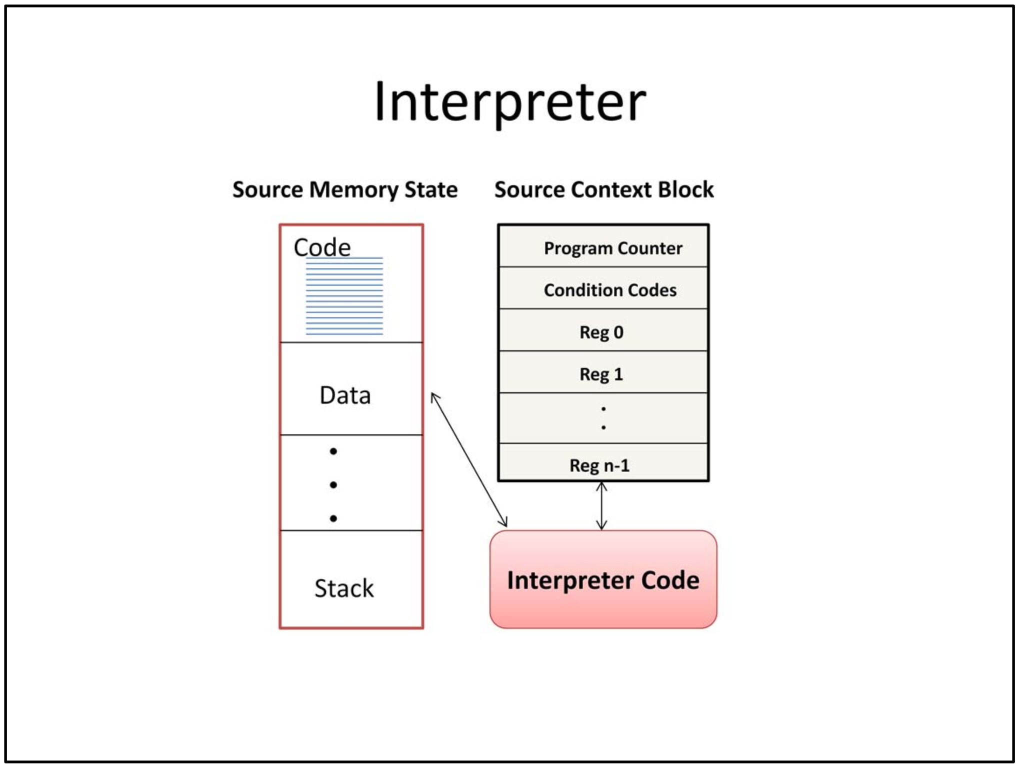


In full virtualization, the Source ISA of a Guest system needs to be mapped to the Target ISA of a Host system. Even if the ISAs are the same, say Intel x86, the Guest and Host system may be in different states and the virtualization software must keep track the Guest state and translate it to the Host state.

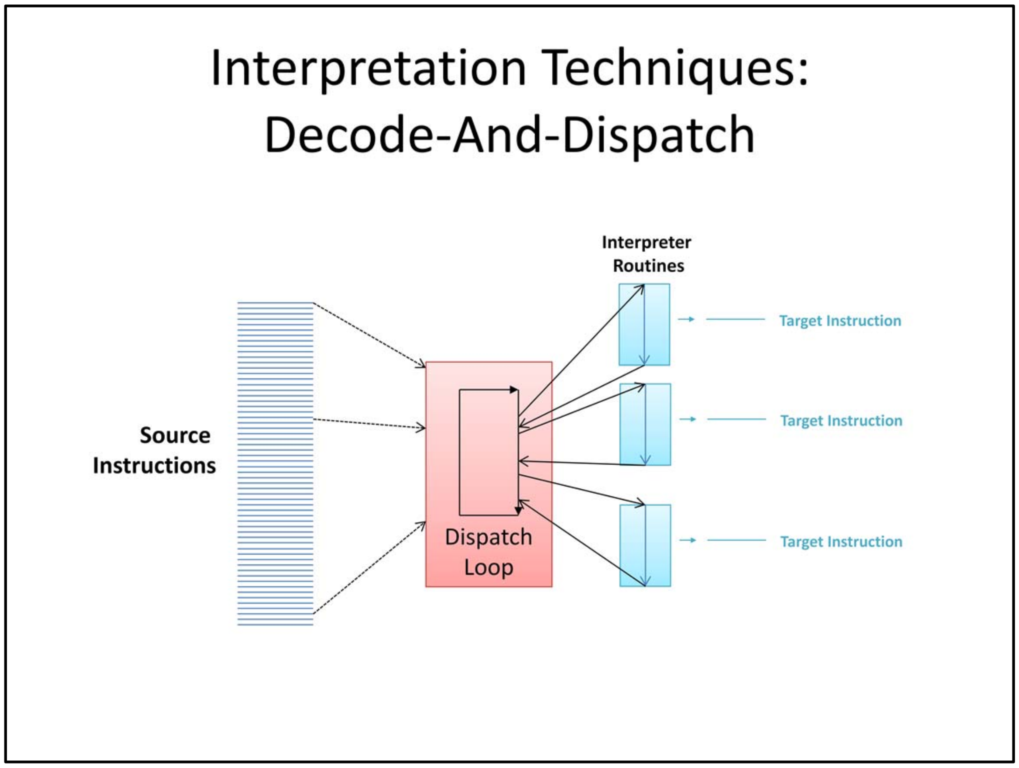


Interpretation

The simplest method to achieve emulation is through interpretation. Each source instruction of the Guest system is fetched, analyzed and performed as an instruction on the Target ISA. This is repeated for every source instruction.

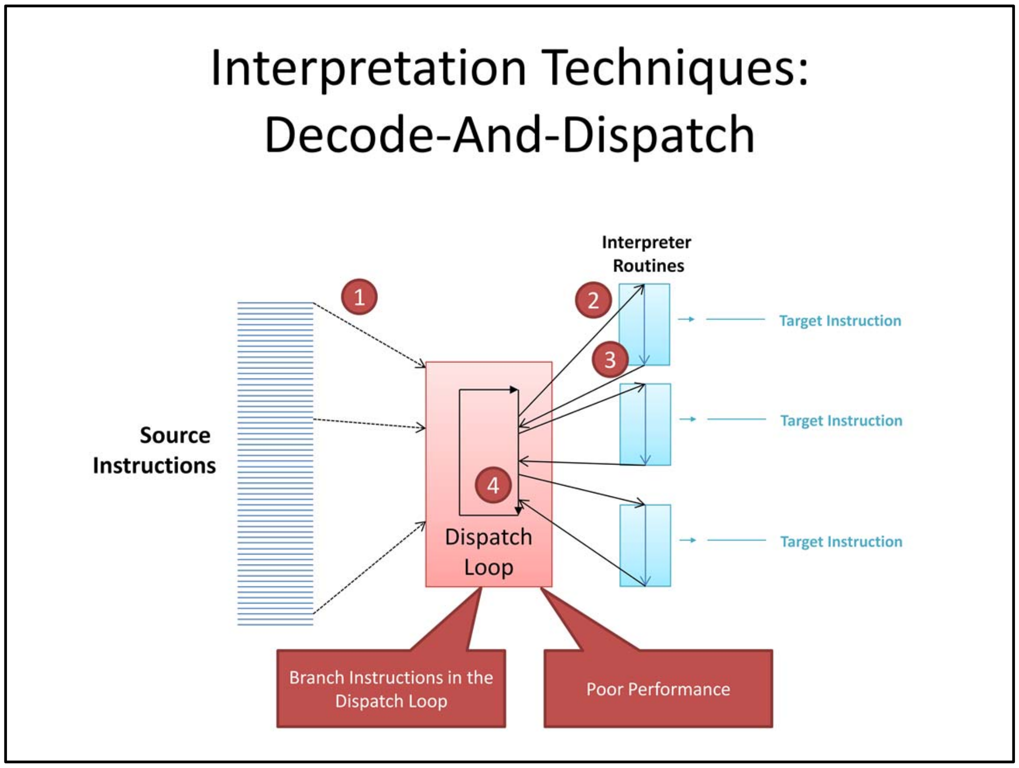

Interpreter

An Interpreter is a program running within the virtualization software that has access to the source memory state as well as the source context block, which allows it to keep track of the guest system’s state and transform it to the necessary instructions on the host system.

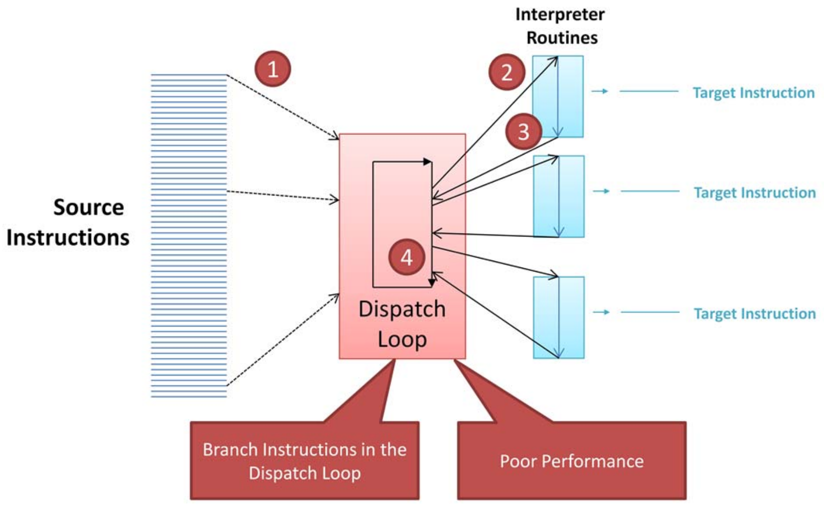


Interpretation Techniques: Decode-And-Dispatch

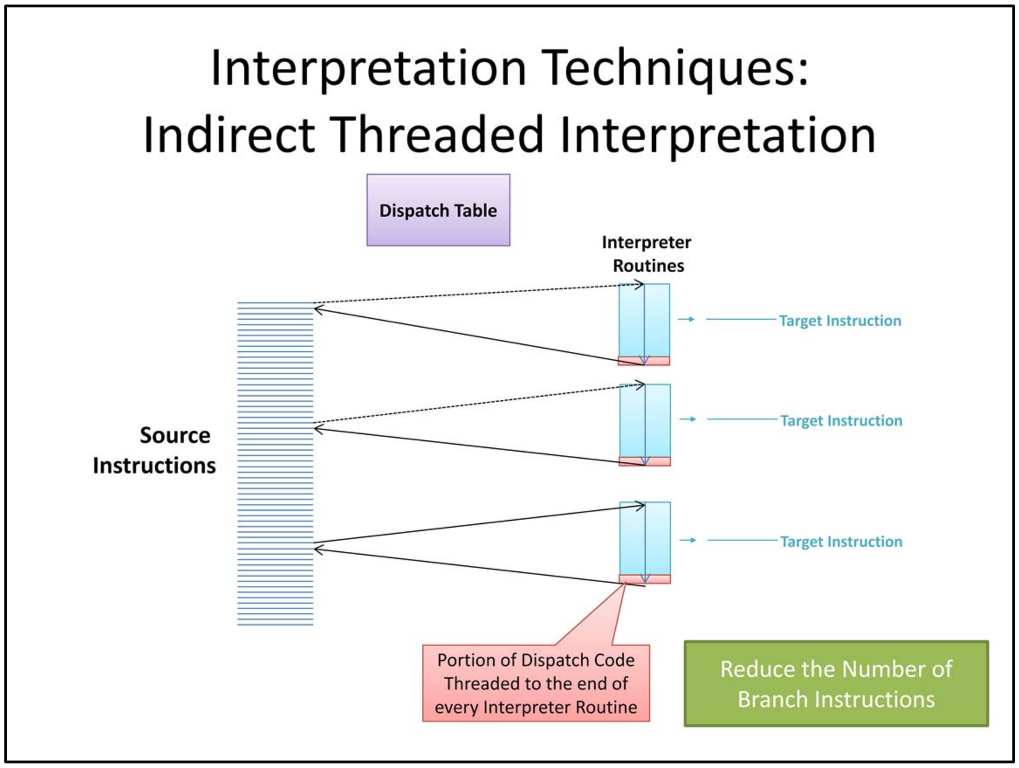
We'll look at a few techniques used to implement interpreters, starting with the decode-and-dispatch technique.
A decode-and-dispatch interpreter steps through the source program instruction by instruction and executes a routine that performs the required translation of the source instruction to the target instruction. The central component of the decode and dispatch interpreter is the dispatch loop, which is basically a switch statement that selects the appropriate interpreter routine for the particular instruction to be interpreted.



Interpretation Techniques: Decode-And-Dispatch



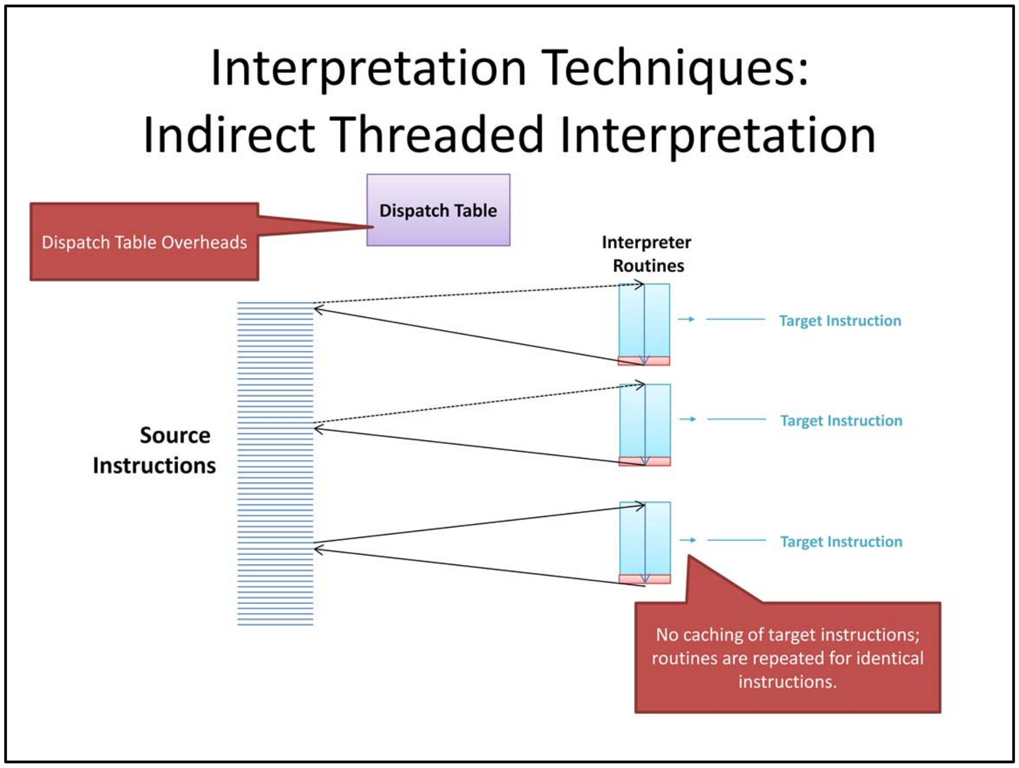
While simple, the decode-and-dispatch technique is not without overheads. The central dispatch loop becomes the bottleneck in the interpretation process as contains a number of branch instructions to select the appropriate interpreter routine. There are at least 4 – an indirect branch for the switch statement, a branch to the required interpreter routine, a second register indirect branch to return the from the interpreter routine and a branch that terminates the loop. These branches lead to poor performance of the interpreter.



Interpretation Techniques: Indirect Threaded Interpretation

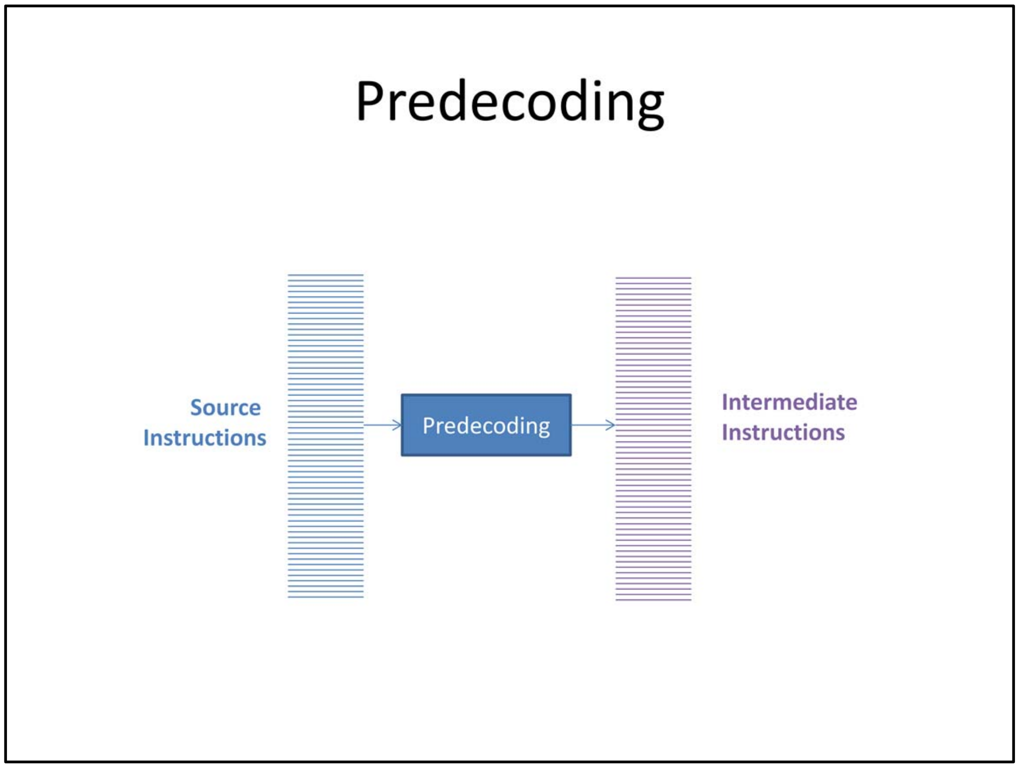
Dispatch Table
Interpreter Routines
Target Instruction
Target Instruction
Target Instruction
Source Instructions
Portion of Dispatch Code Threaded to the end of every Interpreter Routine
Reduce the Number of Branch Instructions

One optimization to the interpreter to improve performance is known as the Indirect threaded interpretation technique. In this technique, a portion of the dispatch code is appended to the end of each interpreter routine. An interpreter routine is located using a dispatch table, and is reached by a jump instruction when stepping through the source program. Through this approach the number of branch instructions can be reduced.

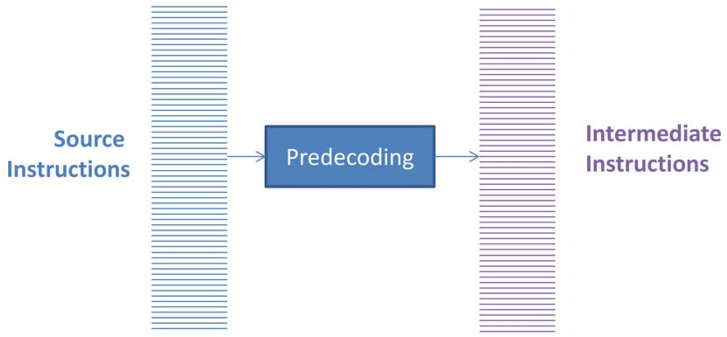


Interpretation Techniques: Indirect Threaded Interpretation

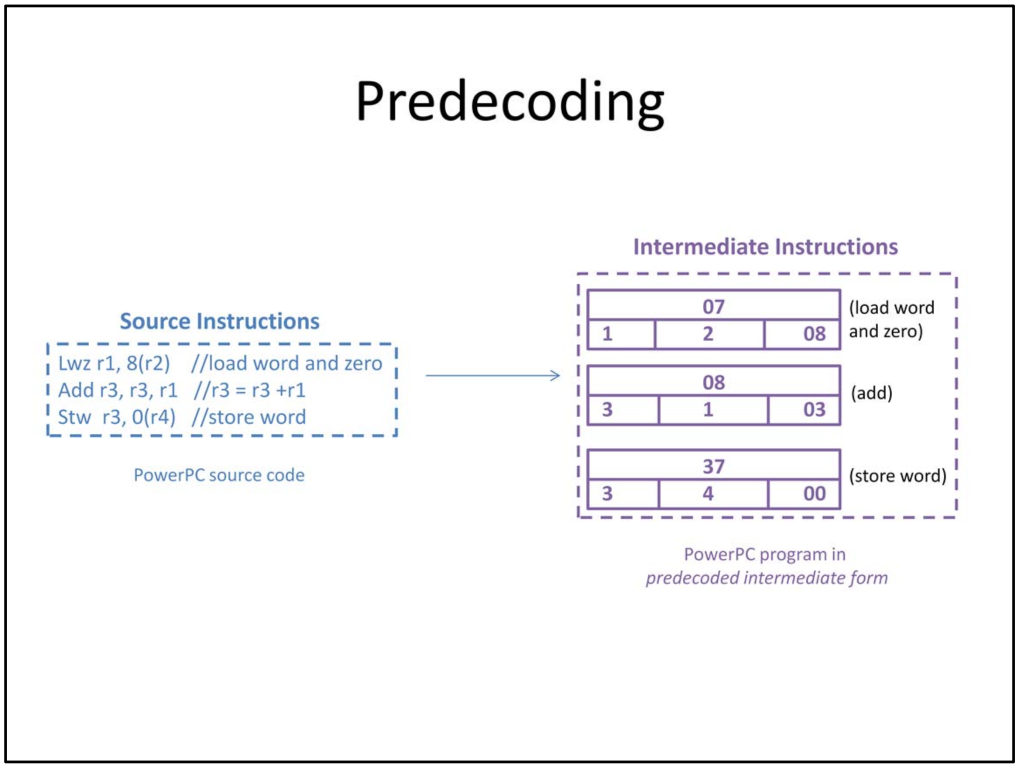
However, there are overheads in this technique. The dispatch table still requires a lookup, and a branch instruction. In addition, there is no caching of interpreted target instructions, so the entire interpretation routine is repeated for every identical instruction in the source.



Predecoding



In order to minimize the effort involved in interpretation of individual source instructions, a technique called predecoding allows for source instructions to be translated to an intermediate form that can be reused when an instruction is re-encountered for emulation. A target program counter is required to step through the intermediate code.

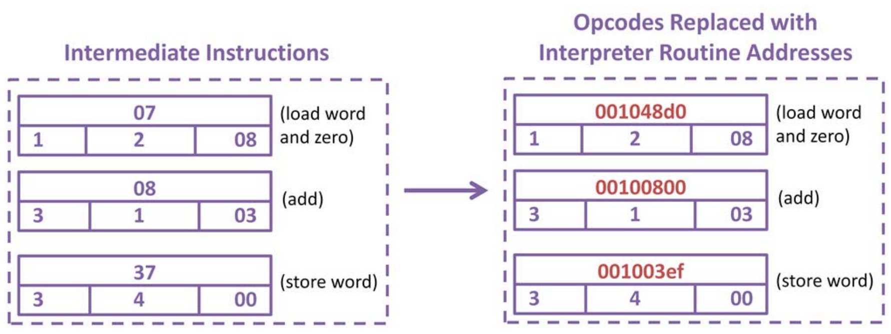


Predecoding

```
Lwz r1, 8(r2)    //load word and zero
Add r3, r3, r1   //r3 = r3 +r1
Stw  r3, 0(r4)   //store word
```

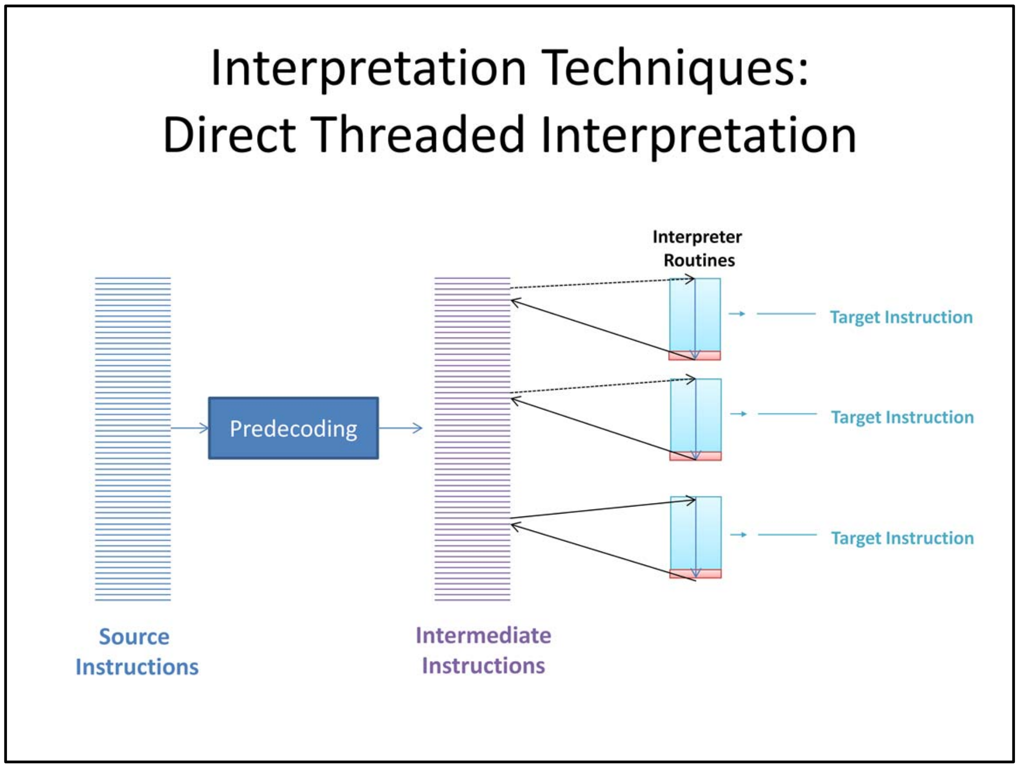
An example of predecoding is on shown here where the source instructions are parsed and converted into an intermediate form that is easier to interpret into the target ISA.

Predecoding



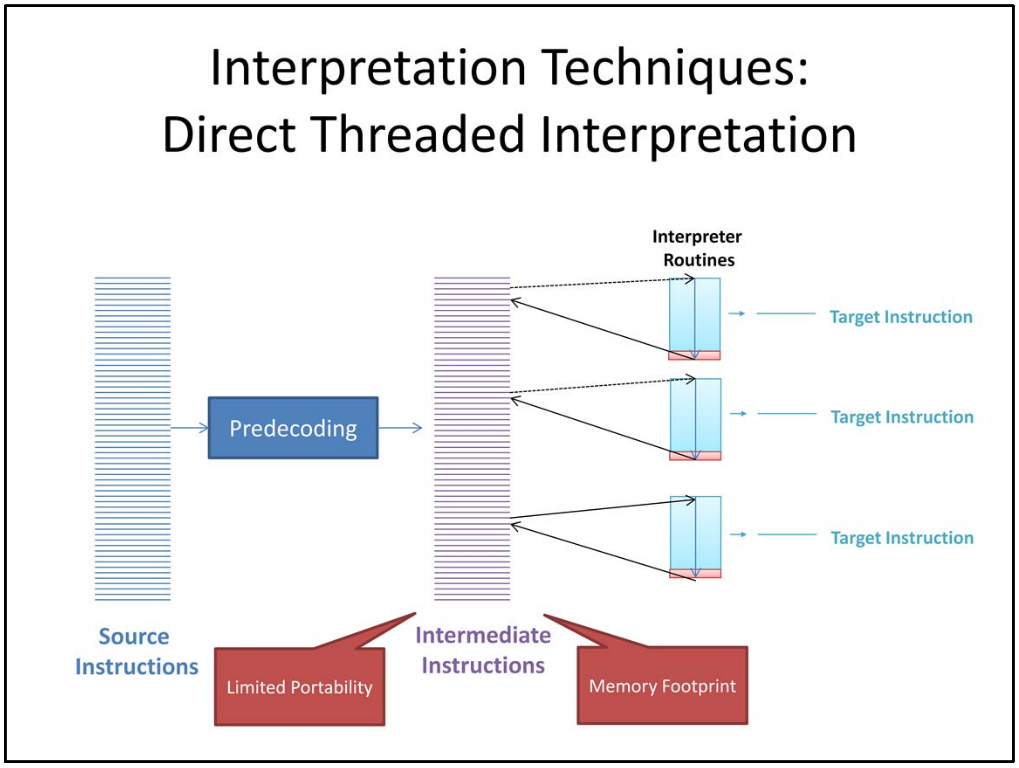
▪To avoid a memory lookup whenever the dispatch table is accessed, the opcode in the intermediate form can be replaced with the address of the interpreter routine

▪This leads to a scheme referred to as direct threaded interpretation



Interpretation Techniques: Direct Threaded Interpretation

This leads to a new interpretation technique known as direct threaded implementation. In this technique the overhead of interpretation is reduced the instruction stream is predecoded and the interpretation routines themselves can be less complex and take less time to execute



Interpretation Techniques: Direct Threaded Interpretation

Primary drawbacks of this technique include limited portability as the intermediate form is dependent on the exact locations of the interpretation routines in memory. This technique also requires a larger memory footprint as the size of the predecoded instructions is proportional to the source memory image. In addition, identical instructions still have to go through an interpretation routine, even if the work involved within the routine is reduced.