

Decoding Machine Instructions

Akash Garg

Problem Statement

- Separating Instructions from Data in binaries
- Decompiling the Binaries
- z80 Binaries
- Using Robby Roto
- Compared results with dZ80 Disassembler

Problems with Decompilation

- Disassembling binaries does not produce unique result
- Infinite source programs from same bit patterns
- Identifying code and data analogous to halting problem
- Cannot determine which instructions will execute (code following loop).
- Problem due to Von Neumann architecture

Identifying Instructions

- Flow-Tracing algorithm presented by Horspool and Marovac
- Restrictions:
 1. No self-modifying code exists in the binary.
 2. Branch addresses are not dynamically computed.
 3. Subroutines return control to their calling points not to their calling points plus some bytes to pass over parameter lists.

Flow-Tracing Algorithm

1. Initialize a stack S to hold the entry point address of the binary. Initialize the map to show all locations in binary as data.
2. If S is empty, then finished.
3. Pop an address A from S .
4. If A is already shown in the map as an instruction, goto step 2.
5. Record location A as holding an instruction in the map.
6. Decode instruction at A .
7. If the instruction is conditional branch or subroutine link, push the destination address on S .
8. If the instruction is stop instruction, subroutine return or unconditional branch, goto step 2.
9. Increment A by the length of the instruction and goto step 4.

Placing Labels

- Each addr put on stack is beginning of block of instructions.
- Each addr that goes back to step 2 is end of block.
- Addresses placed on stack are locations for labels.
- Create control flow graphs using this method.

Identifying Data

- Two approaches:
 1. The data item format can only be representative of a known format.
 2. The semantics of the instructions where the data item is used indicate the appropriate format.
- Make pass through instructions to determine data operands.

Binaries Used

Addr. Range	Binary File
0000 - 0fff	rotox1.bin
1000 - 1fff	rotox2.bin
2000 - 2fff	rotox3.bin
3000 - 3fff	rotox4.bin
4000 - 7fff	RAM (blank)
8000 - 8fff	rotox5.bin
9000 - 9fff	rotox6.bin
a000 - afff	rotox7.bin
b000 - bfff	rotox8.bin
c000 - cfff	rotox9.bin
d000 - dfff	rotox10.bin
e000 - e7ff	NVRAM (blank)
e800 - ffff	RAM (blank)

Comparitive Results

Binary	Flow Tracing	dZ80
rotox1.bin	79	2730
rotox2.bin	320	2220
rotox3.bin	2	3137
rotox4.bin	11	2729
rotox5.bin	132	2759
rotox6.bin	1	2615
rotox7.bin	12	2858
rotox8.bin	13	3017
rotox9.bin	533	3009
rotox10.bin	127	3020

Extreme Cases

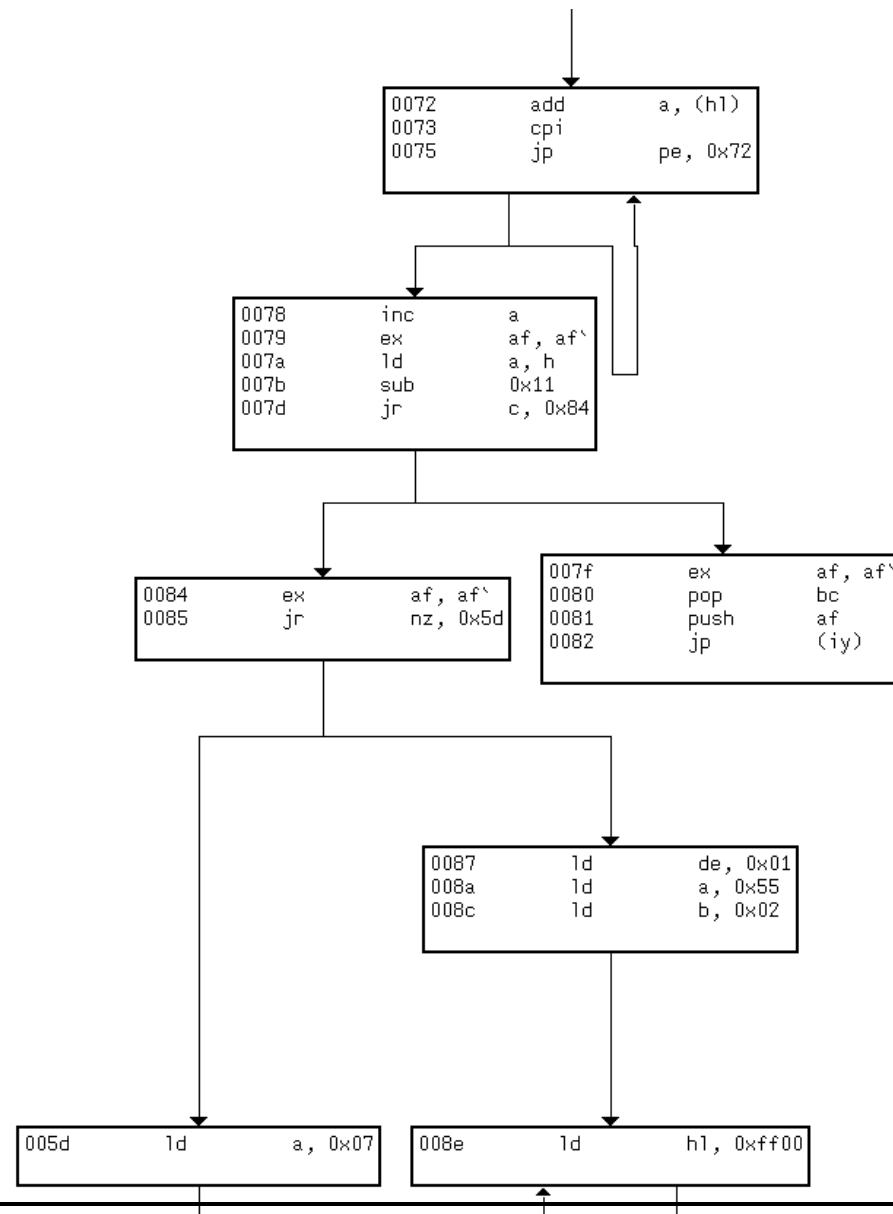
Decoding rotox3.bin:

```
0000    rr    e
0002    halt
```

Decoding rotox6.bin:

```
0000    ret
```

Dynamic Branching



Improvements

- Basic flow tracing places strong restrictions on binary.

```
Y = 10 ;  
X = Y ;  
X = X + 2 ;  
JUMP (X) ;
```

- Def-Use chains
- Iterative execution while decompiling

```
Read (X) ;  
Y = F (X) ;  
JUMP (Y) ;
```

- Still possible to miss some instructions
- Value-set Analysis