CHIP-8 Static Recompiler Using LLVM

Luke Sheeran
Pembroke College

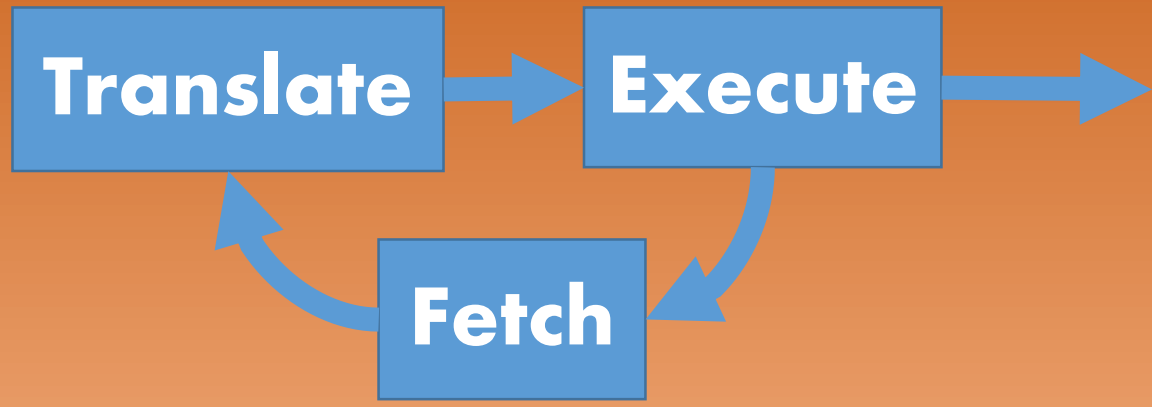

AVI
COSMAC VIP
10 AMP
Panasonic
Weller

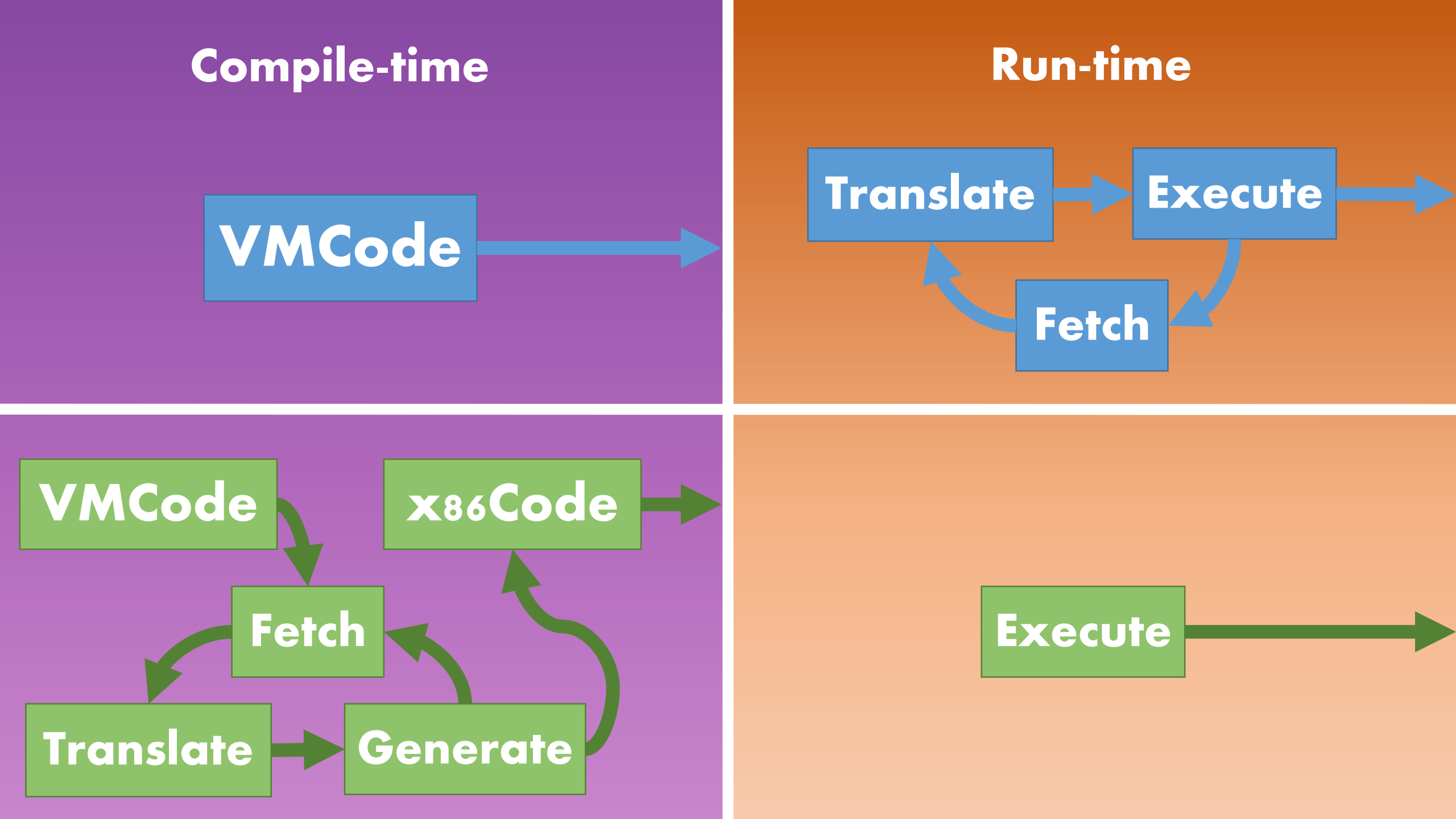
Compile-time

Run-time

Compile-time

Run-time

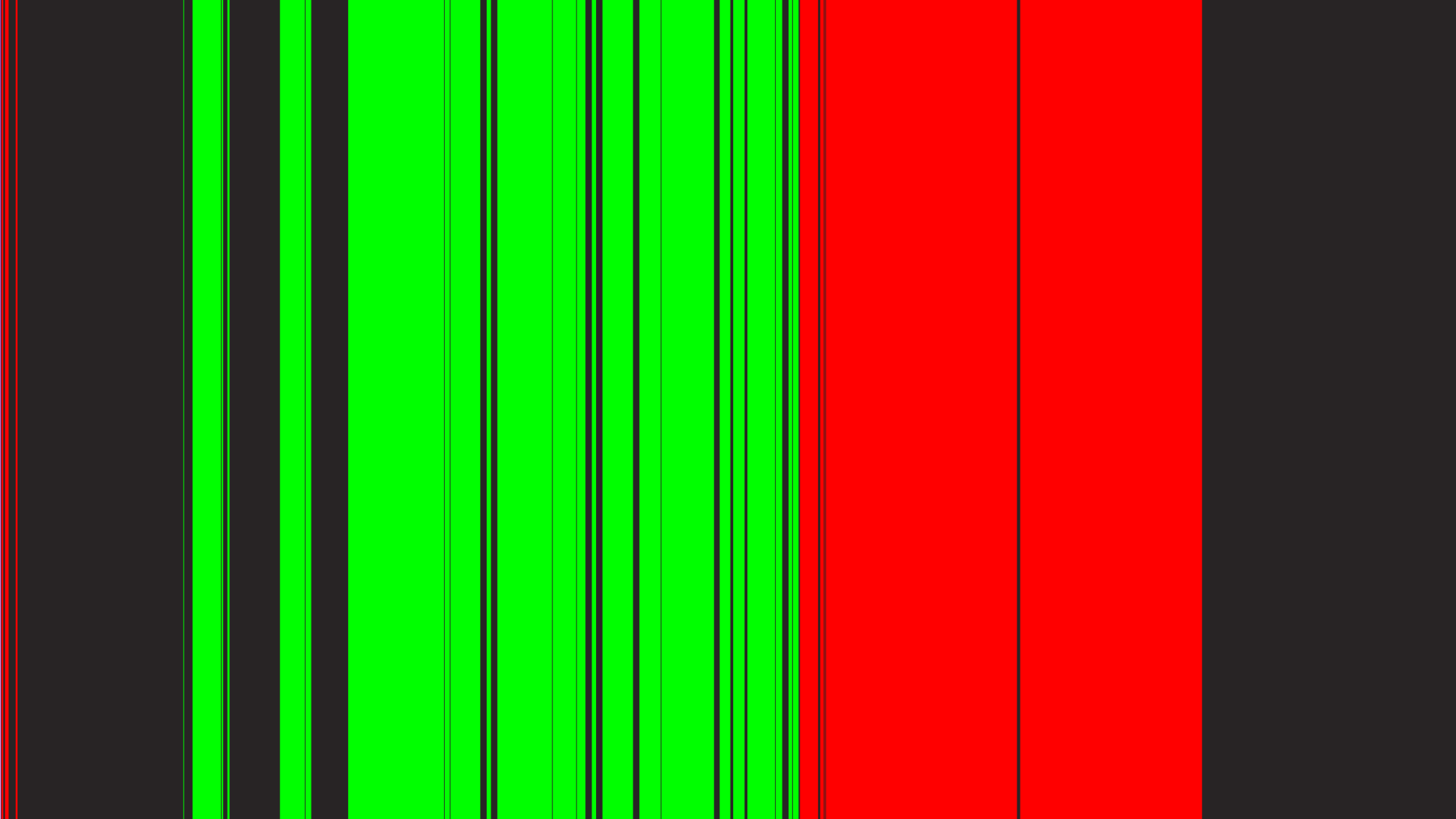


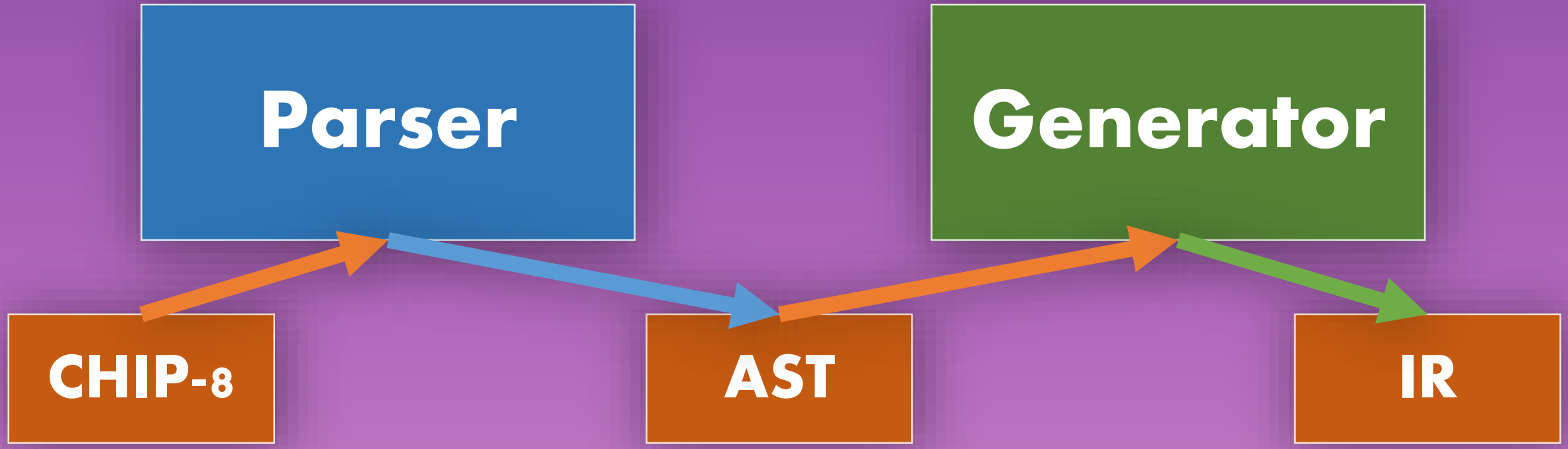

Compile-time
VMCode
Run-time
Translate
Execute
Fetch
VMCode
x86Code
Fetch
Translate
Generate
Execute

Assembler

Memory Trace

Custom c8ROMs




Parser
Generator
CHIP-8
AST
IR

8XY4

```
auto valX = t->builder.CreateLoad(t->builder.getInt8Ty(), t->rX[t->currentNode->operands[0]/*X*/]);
auto zExtX16 = t->builder.CreateZExt(valX, t->builder.getInt16Ty());

auto valY = t->builder.CreateLoad(t->builder.getInt8Ty(), t->rX[t->currentNode->operands[1]/*Y*/]);
auto zExtY16 = t->builder.CreateZExt(valY, t->builder.getInt16Ty());

auto addXnY16 = t->builder.CreateAdd(zExtX16, zExtY16); //add zero extended registers together

auto byte = llvm::ConstantInt::get(t->mainModule->getContext(), llvm::APInt(16, 255));
auto oflw = t->builder.CreateICmpUGT(addXnY16, byte); //check if 16bit add is over 0xFF
auto zExtOflw = t->builder.CreateZExt(oflw, t->builder.getInt8Ty());
t->builder.CreateStore(zExtOflw, t->rX[15/*F*/]); //convert i1 bool to i8 and store it as overflow flag
auto addXnY = t->builder.CreateTrunc(addXnY16, t->builder.getInt8Ty());
t->builder.CreateStore(addXnY, t->rX[t->currentNode->operands[0]/*X*/]); //truncate and store
```

00BX	Debug	Print register
0NNN	Call	
00E0	Display	disp_clear()
00EE	Flow	return;
1NNN	Flow	goto NNN;
2NNN	Flow	*(0xNNN)()
3XNN	Cond	if(Vx==NN)
4XNN	Cond	if(Vx!=NN)
5XY0	Cond	if(Vx==Vy)
6XNN	Const	V*x* = NN
7XNN	Const	V*x* += NN
8XY0	Assign	V*x*=Vy
8XY1	BitOp	Vx=V*x*|V*y*
8XY2	BitOp	Vx=V*x*&V*y*
8XY3	BitOp	Vx=Vx^Vy
8XY4	Math	Vx += Vy
8XY5	Math	Vx -= Vy
8XY6	BitOp	Vx >> 1
8XY7	Math	Vx=Vy-Vx
8XYE	BitOp	Vx << 1
9XY0	Cond	if(Vx!=Vy)

ANNN	MEM	I = NNN
BNNN	Flow	PC=V0+NNN
CXNN	Rand	Vx=rand()&NN
DXYN	Disp	draw(Vx,Vy,N)
EX9E	KeyOp	if(key()==Vx)
EXA1	KeyOp	if(key()!=Vx)
FX07	Timer	Vx = get_delay()
FX0A	KeyOp	Vx = get_key()
FX15	Timer	delay_timer(Vx)
FX18	Sound	sound_timer(Vx)
FX1E	MEM	I +=Vx
FX29	MEM	I=sprite_addr[Vx]
FX33	BCD	set_BCD(Vx); *(I+0)=BCD(3); *(I+1)=BCD(2); *(I+2)=BCD(1);
FX55	MEM	reg_dump(Vx,&I)
FX65	MEM	reg_load(Vx,&I)

```
1  7001 //add 1 to r0
2  8104 //get r1 = 1 using 8XY4
3  8211 //get r2 = 1 using 8XY1
4  801E //bitshift left r0
5  801E
6  801E
7  8023 //xor to set bit
8  801E
9  801E
10 801E
11 00B0 //print 'H'
12 812E //bitshift left r1
13 812E
14 812E
15 812E
16 812E
17 8124 //add to set bit
18 8210 //copy r1 to r2
19 8101 //or r2 with r0
20 00B1 //print 'i'
21 00B2 //print '!'
```

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   70 01 81 04 82 11 80 1E 80 1E 80 1E 80 23 80 1E  p...,.€.€.€.€#€.
00000010   80 1E 80 1E 00 B0 81 2E 81 2E 81 2E 81 2E 81 2E  €.€..°..........
00000020   81 24 82 10 81 01 00 B1 00 B2                    .$,....±.²
```

Assembler

Generator

```
define i32 @main() {
entry:
  %0 = load i8, i8* @r0
  %1 = add i8 %0, 1
  store i8 %1, i8* @r0
  %2 = load i8, i8* @r1
  %3 = zext i8 %2 to i16
  %4 = load i8, i8* @r0
  %5 = zext i8 %4 to i16
  %6 = add i16 %3, %5
  %7 = icmp ugt i16 %6, 255
  %8 = zext i1 %7 to i8
  store i8 %8, i8* @rF
  %9 = trunc i16 %6 to i8
  store i8 %9, i8* @r1
  %10 = load i8, i8* @r2
```

```
/home/ls/Documents/Projects/llvm/llvm-7.0.1.src/build/bin/lli c8test.ll
Hi![ls@sticky Dissertation]$
```

Compiler