Language Implementation Basics

Polyglot Programming DC 2015

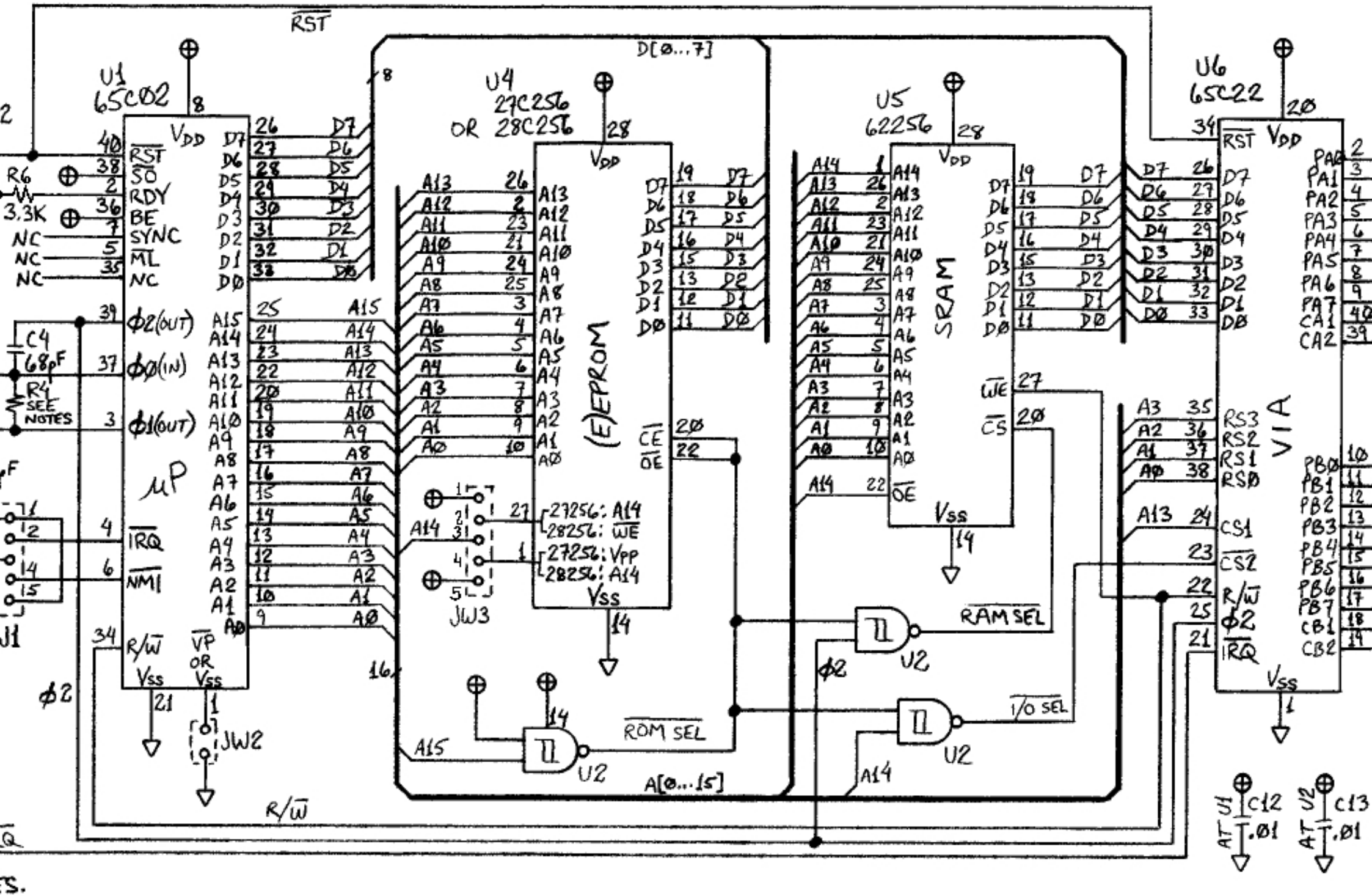
Brock Wilcox
awwaiid@thelackthereof.org
bwilcox@optoro.com

Full Stack.

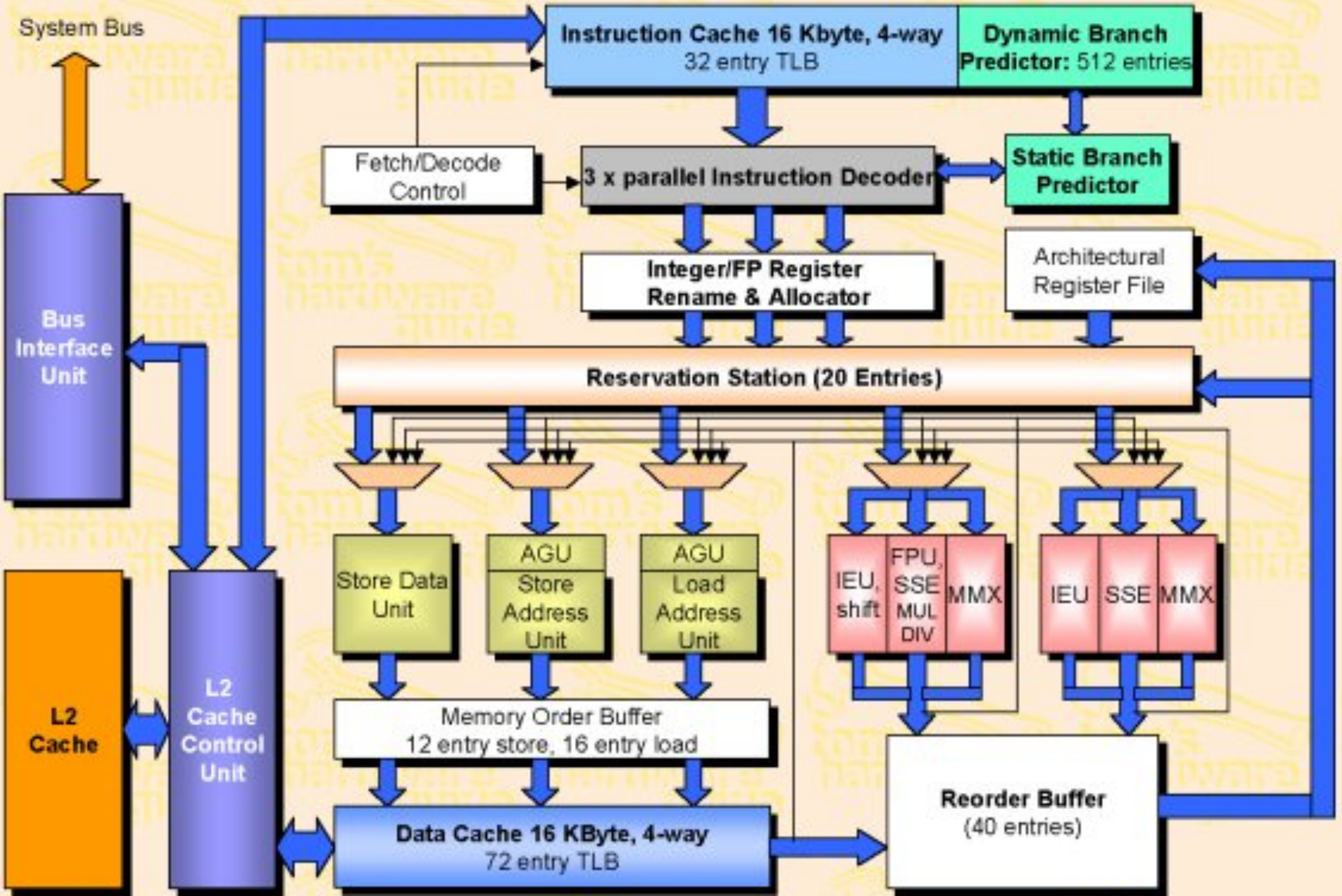
Pft.

Transistors → Logic Gates → Logic Units

* Fetch contents of memory
* CASE statement of what to do next
* Local storage (registers)


RST
D[0...7]
U1
65C02
VDD
RST
SO
RDY
BE
SYNC
ML
NC
NC
NC
Φ2(OUT)
ΦØ(IN)
Φ1(OUT)
IRQ
NMI
R/W
VP OR Vss
Vss
μP
R6
3.3K
C4
68pF
R4
SEE NOTES
JW1
JW2
Φ2
R/W
U4
27C256
OR 28C256
(E)EPROM
CE
OE
27256: A14
28256: WE
27256: Vpp
28256: A14
JW3
U5
62256
SRAM
WE
CS
OE
U6
65C22
VIA
RS3
RS2
RS1
RSØ
CS1
CS2
R/W
Φ2
IRQ
PAØ
PA1
PA2
PA3
PA4
PA5
PA6
PA7
CA1
CA2
PBØ
PB1
PB2
PB3
PB4
PB5
PB6
PB7
CB1
CB2
U2
RAM SEL
I/O SEL
ROM SEL
A15
A14
A[0...15]
U1 AT C12 .01
U2 AT C13 .01

Pentium(r) III Processor Architectural Block Diagram



Input is binary things in memory.

Output is binary things in memory.

```
0000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 3e00 0100 0000 b000 4000 0000 0000  ..>.......@.....
0000020: 4000 0000 0000 0000 0001 0000 0000 0000  @...............
0000030: 0000 0000 4000 3800 0200 4000 0400 0300  ....@.8...@.....
0000040: 0100 0000 0500 0000 0000 0000 0000 0000  ................
0000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
0000060: d200 0000 0000 0000 d200 0000 0000 0000  ................
0000070: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
0000080: d400 0000 0000 0000 d400 6000 0000 0000  ..........`.....
0000090: d400 6000 0000 0000 0d00 0000 0000 0000  ..`.............
00000a0: 0d00 0000 0000 0000 0000 2000 0000 0000  .......... .....
00000b0: b804 0000 00bb 0100 0000 b9d4 0060 00ba  ............`...
00000c0: 0d00 0000 cd80 b801 0000 00bb 0000 0000  ................
00000d0: cd80 0000 4865 6c6c 6f20 776f 726c 6421  ....Hello world!
00000e0: 0a00 2e73 6873 7472 7461 6200 2e74 6578  ...shstrtab..tex
00000f0: 7400 2e64 6174 6100 0000 0000 0000 0000  t..data.........
0000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000140: 0b00 0000 0100 0000 0600 0000 0000 0000  ................
0000150: b000 4000 0000 0000 b000 0000 0000 0000  ..@.............
0000160: 2200 0000 0000 0000 0000 0000 0000 0000  "...............
0000170: 1000 0000 0000 0000 0000 0000 0000 0000  ................
0000180: 1100 0000 0100 0000 0300 0000 0000 0000  ................
0000190: d400 6000 0000 0000 d400 0000 0000 0000  ..`.............
00001a0: 0d00 0000 0000 0000 0000 0000 0000 0000  ................
00001b0: 0400 0000 0000 0000 0000 0000 0000 0000  ................
00001c0: 0100 0000 0300 0000 0000 0000 0000 0000  ................
00001d0: 0000 0000 0000 0000 e100 0000 0000 0000  ................
00001e0: 1700 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0100 0000 0000 0000 0000 0000 0000 0000  ................
```

Assembler:

Assembly text -> binary

```
section .data
  hello: db 'Hello world!',0x0A
  helloLen: equ $-hello

section .text
  global _start

_start:
  mov eax, 4              ; Output
  mov ebx, 1              ; to stream 1, STDOUT
  mov ecx, hello          ; pointer
  mov edx, helloLen       ; len
  int 80h                 ; DO IT

  mov eax, 1              ; exit
  mov ebx, 0              ; status 0
  int 80h                 ; DO IT
```

Compiler:

"High Level Language" (HLL) -> binary

Interpreter:

Immediately execute instructions

Interpreted vs Compiled

Typical compiler pieces:

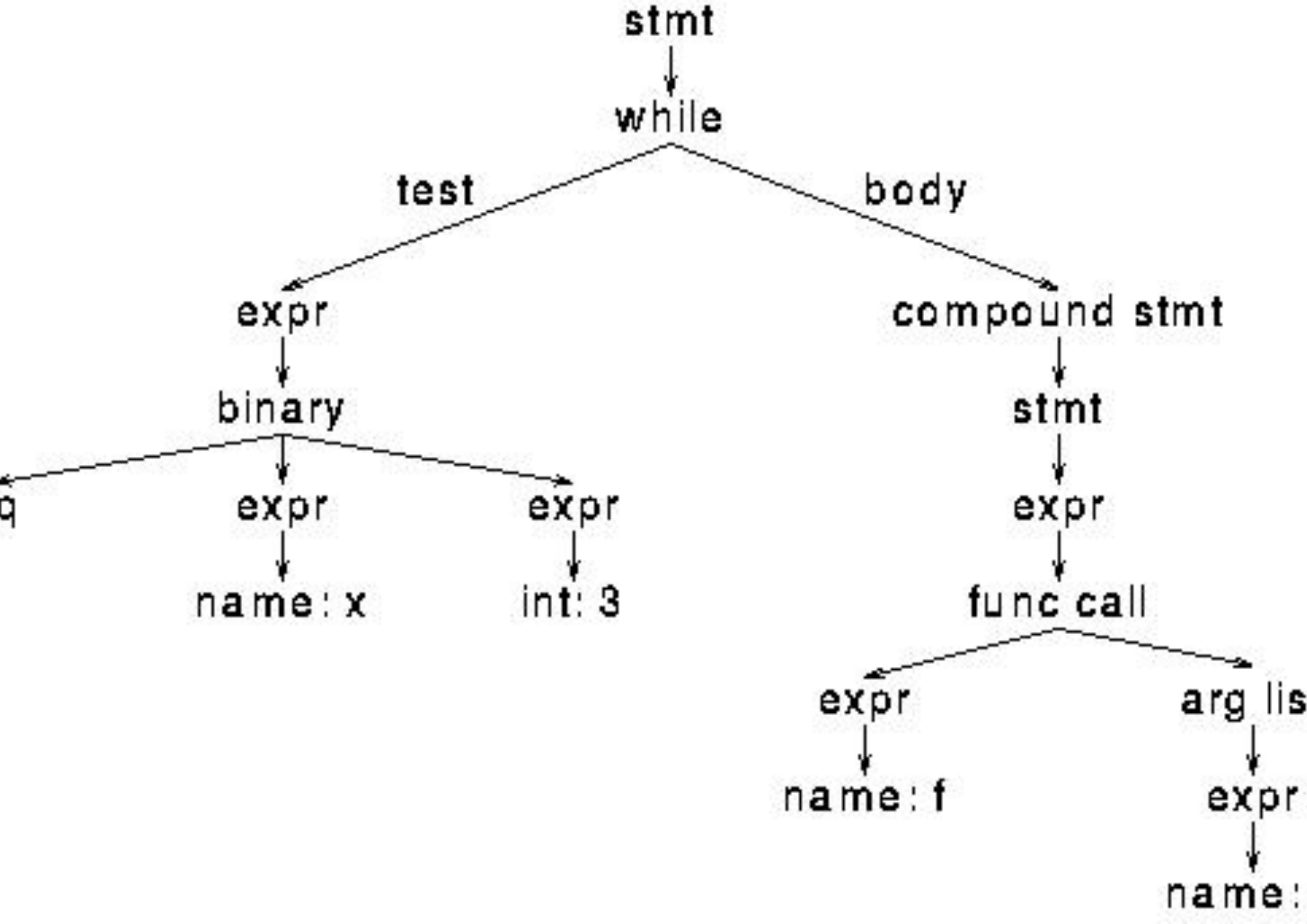
* Lexical Analyzer
* Parser
* AST Builder / Optimizer
* Emitter
* Run-time support

Lexer

characters → words

Parser / AST

words → tree


stmt
while
test
body
expr
compound stmt
binary
stmt
q
expr
expr
expr
name: x
int: 3
func call
expr
arg lis
name: f
expr
name:

Emitter

tree → destination language

Run-time support

* Get things started
* Keep track of variables / memory
* Bridge to outside world

Bytecode Systems

Compile:
Lex/Parse → AST → Bytecode

Interpret:
Bytecode → Data Structures → interpreted+compiled blobs

So!

Let's write an Interpreter

First: The run time

Next: The parser

Finally: Execute!

Well, that was fun.

Let's write a Compiler!