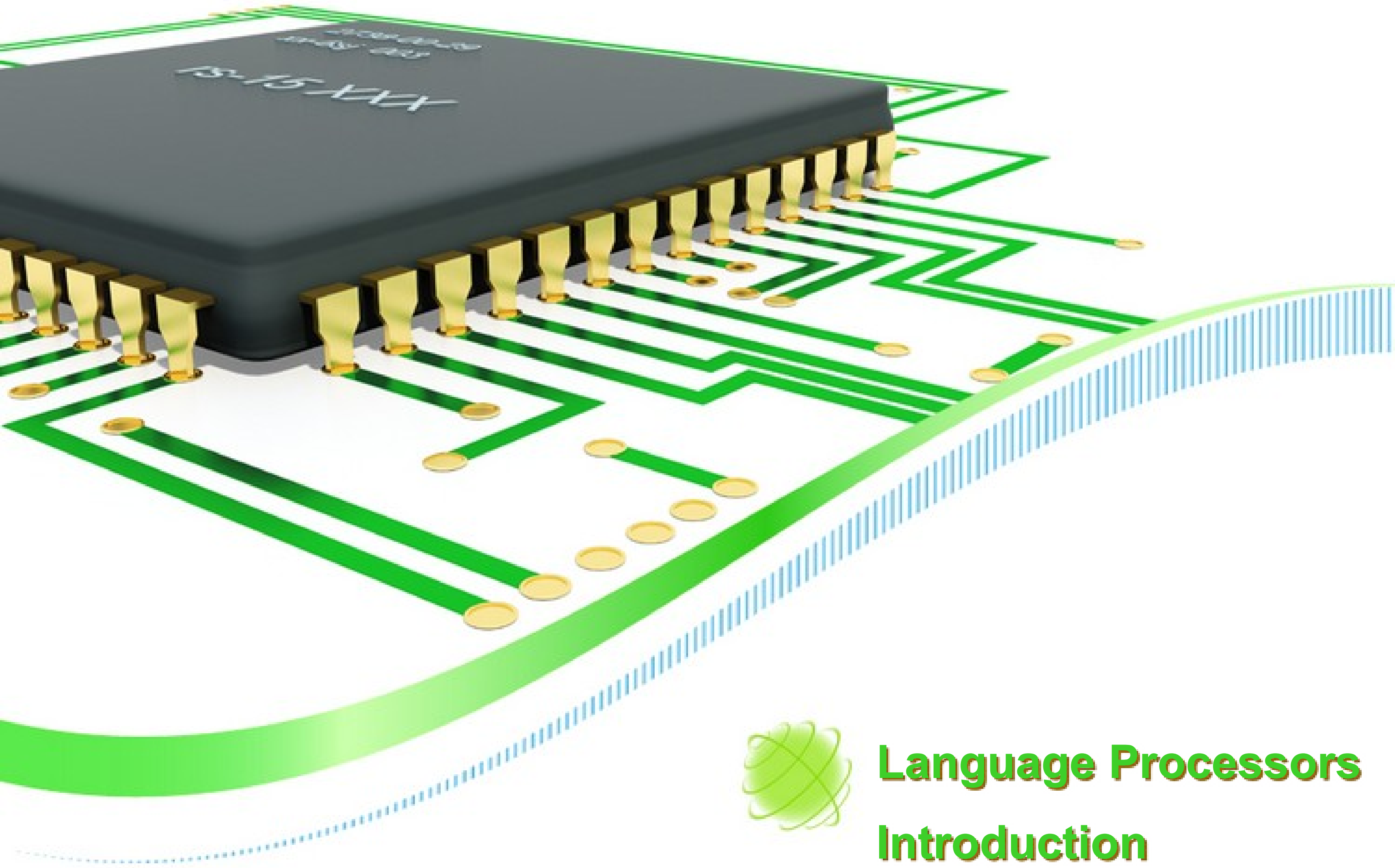
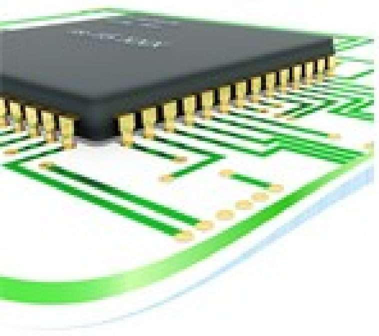


Language Processors
Introduction

Definition



A language processor is a tool that translates formal inputs, which conform to a language, to target representations intended.

Widely Referred Types

Compiler

Simply stated, a compiler is a program that can read a program in one language – the source language – and translate it into an equivalent program in another language – the target language.

Aho, A.V, Ullman J.D, Sethi R., Lam M.S; Dragon Book

A compiler is simply a computer program that translates other computer programs to prepare them for execution.

Cooper, K.D., Torczon, L.; Engineering A Compiler

Interpreter

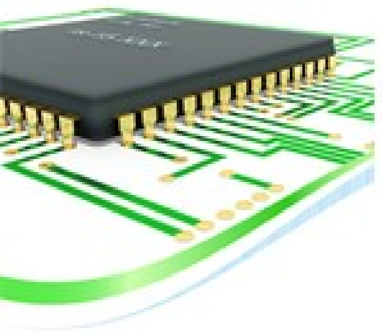
An interpreter is another common kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.

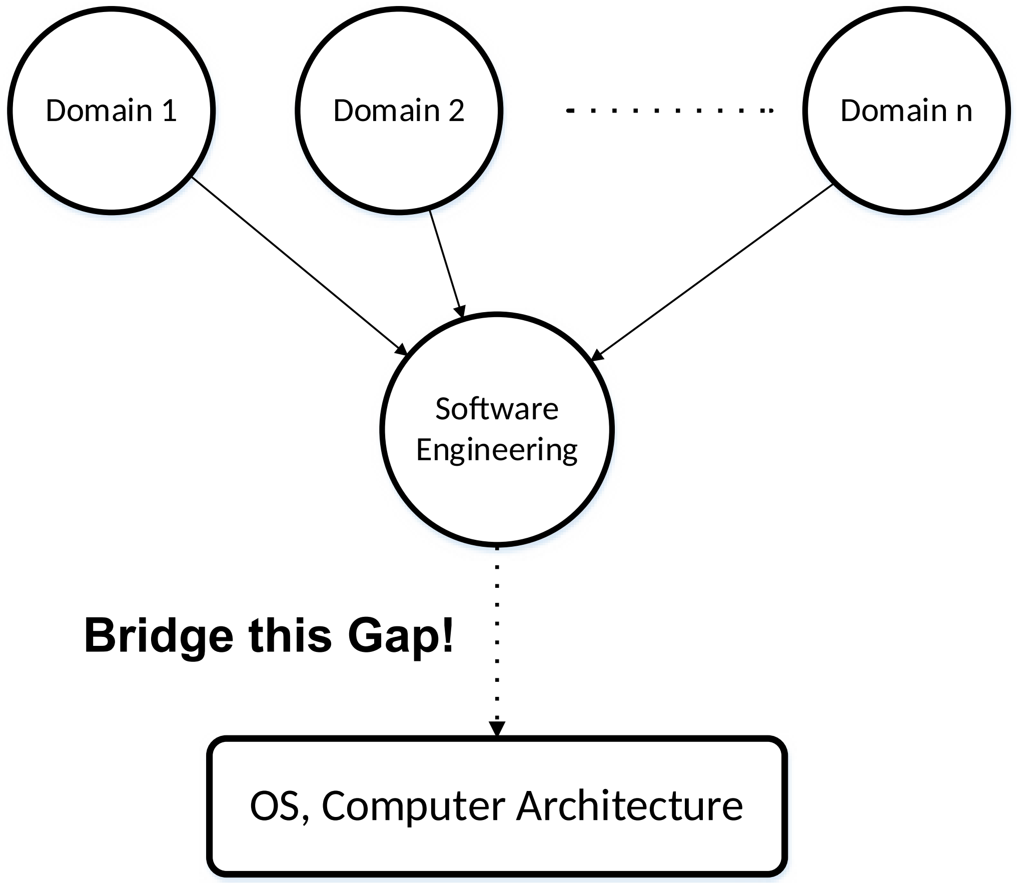
Aho, A.V, Ullman J.D, Sethi R., Lam M.S; Dragon Book

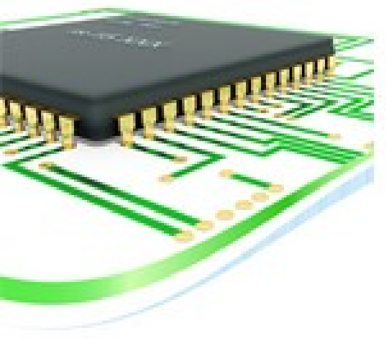
An Interpreter takes as input an executable specification and produces as output the result of executing the specification.

Cooper, K.D., Torczon, L.; Engineering A Compiler

The Need




Domain 1
Domain 2
Domain n
Software Engineering
Bridge this Gap!
OS, Computer Architecture



Yet Another One?

New languages, new language processors…

Do we really need to invent a new one?

Reasons

- Evolutionary
- Technical
- Practical
- Commercial
- …

400+ imperative languages noted so far.

https://en.wikipedia.org/wiki/Timeline_of_programming_languages
Latest retrieval Feb 2nd, 2024

Yet Another One?

New languages, new language processors…

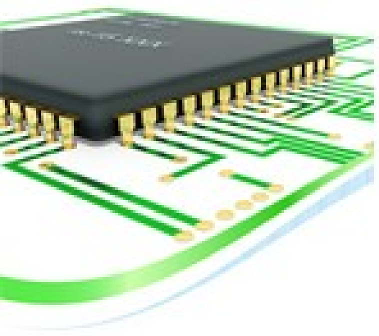
Do we really need to invent a new one?

Reasons

- **… and even political-economic**

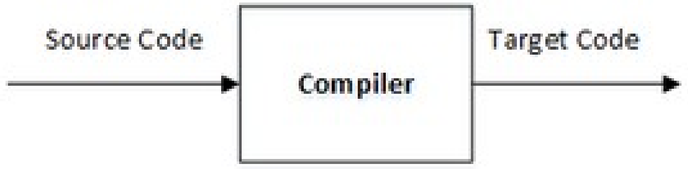
"Much of the past Soviet systems software effort has been in programming languages. This is reflected in the large proportion of the open publications devoted to this area, and is consistent with the given hardware constraints, the relatively formal academic orientation of Soviet software research personnel, and the historical pattern followed in the West. Something like 50 different higher level languages can be identified from the literature. Many are experimental and have had virtually no impact beyond their development groups."

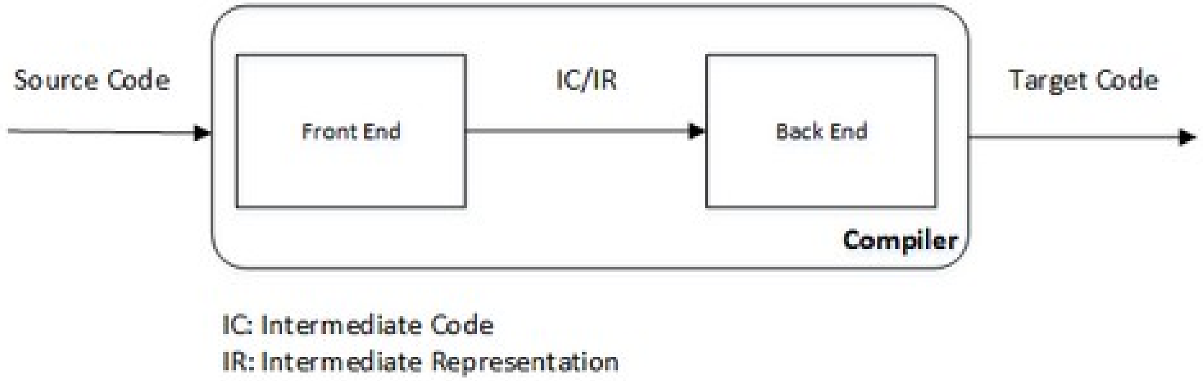
Goodman, S. E. (1979). Software in the Soviet Union: Progress and Problems. Advances in Computers Volume 18, 231–287. doi:10.1016/s0065-2458(08)60585-9



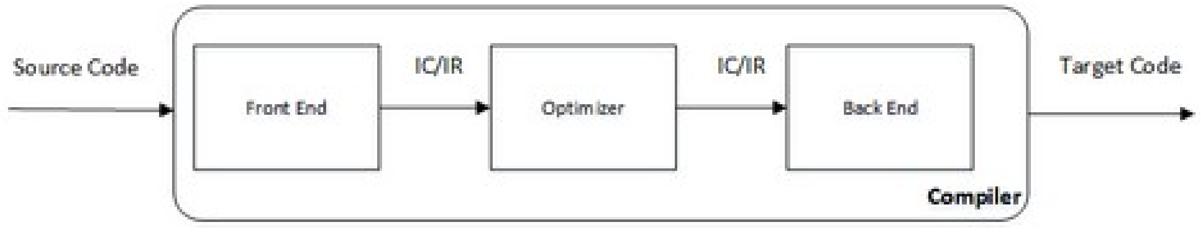
What's in the Box?

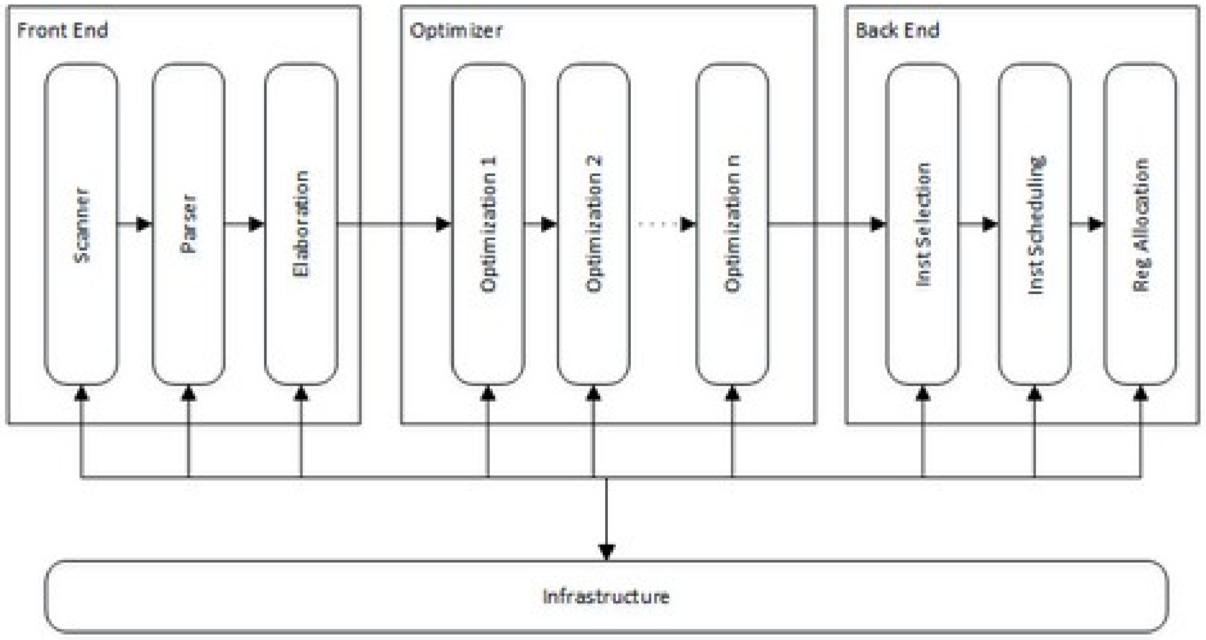
Compiler as an example ...
Details according to the need and taste.



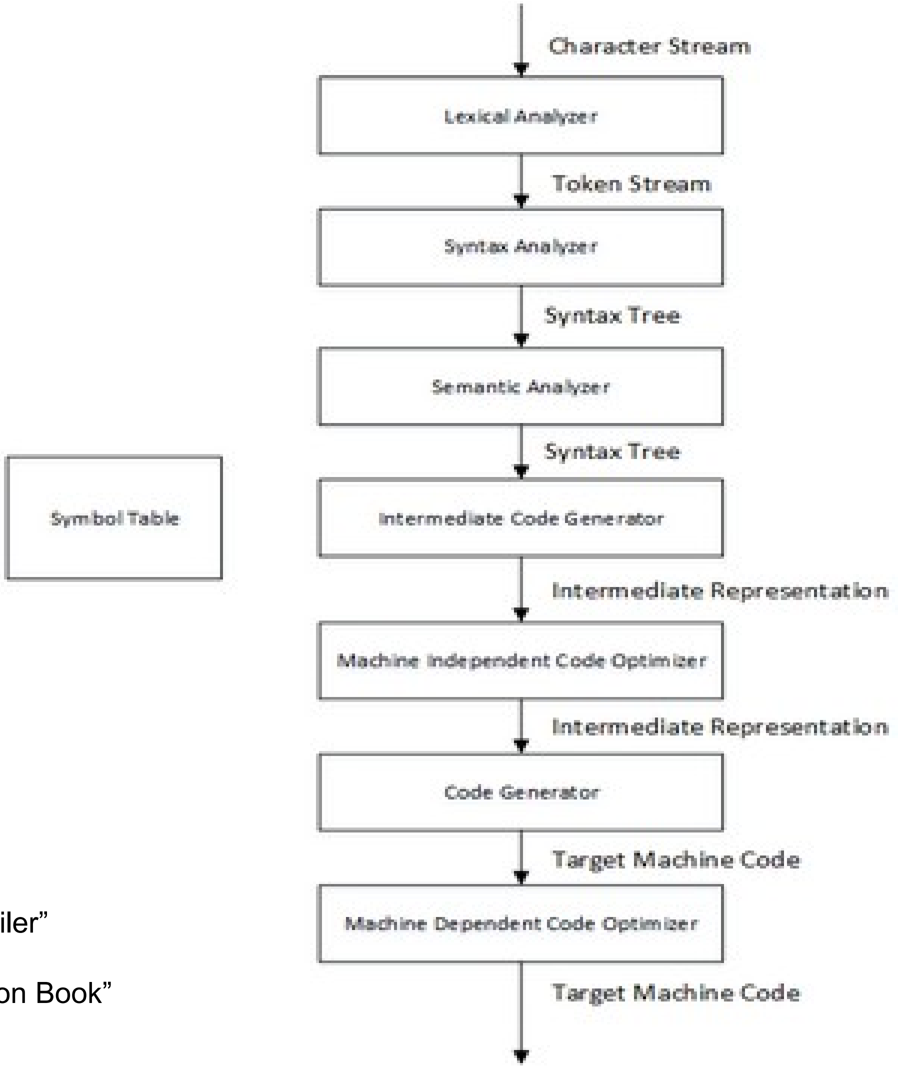


What's in the Box?





What's in the Box?



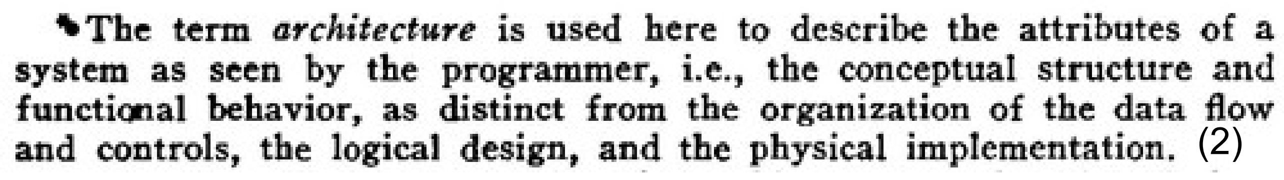
Diagrams are from

"Cooper, K.D., Torczon, L.; Engineering A Compiler"

"Aho, A.V, Ullman J.D, Sethi R., Lam M.S; Dragon Book"

Architecture

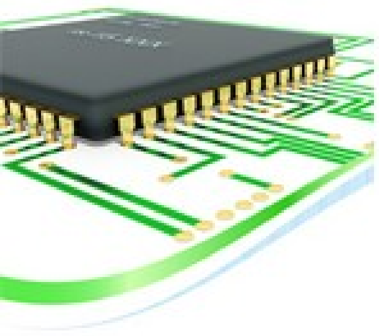
- Programmers see the computer through the window provided by the designers at the level of its functional architecture.
- This window is provided by you, the designer. (1)



The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation. (2)

1. From Prof. Dr. Bozşahin's CENG444 course lecture.
2. Amdahl, Gene & Blaauw, Gerrit & Brooks, Jr, Frederick. (2000). Architecture of the IBM System/360. IBM Journal of Research and Development. 44. 21-36. 10.1147/rd.82.0087.

Architecture

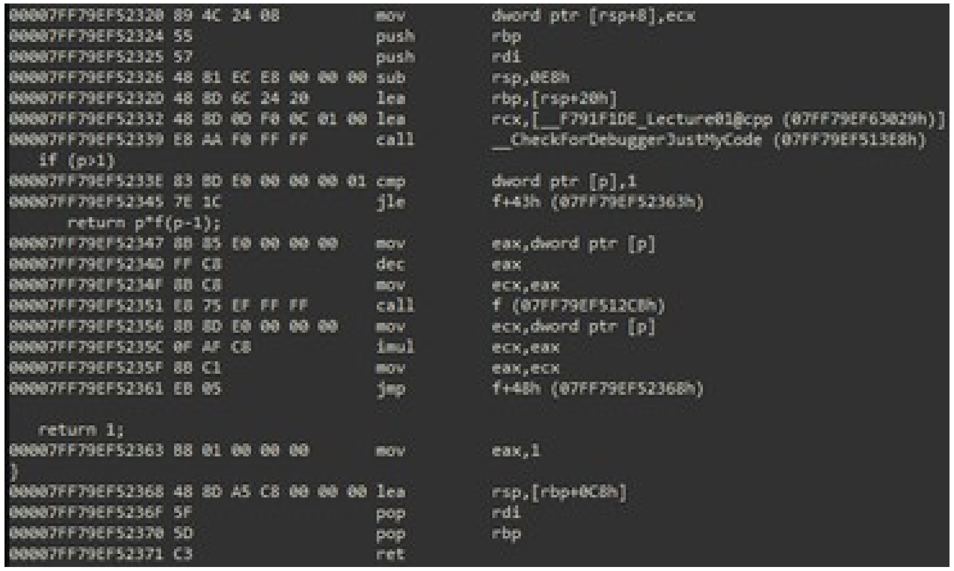


```
int f(int p)
{
   if (p>1)
      return p*f(p-1);

   return 1;
}
```

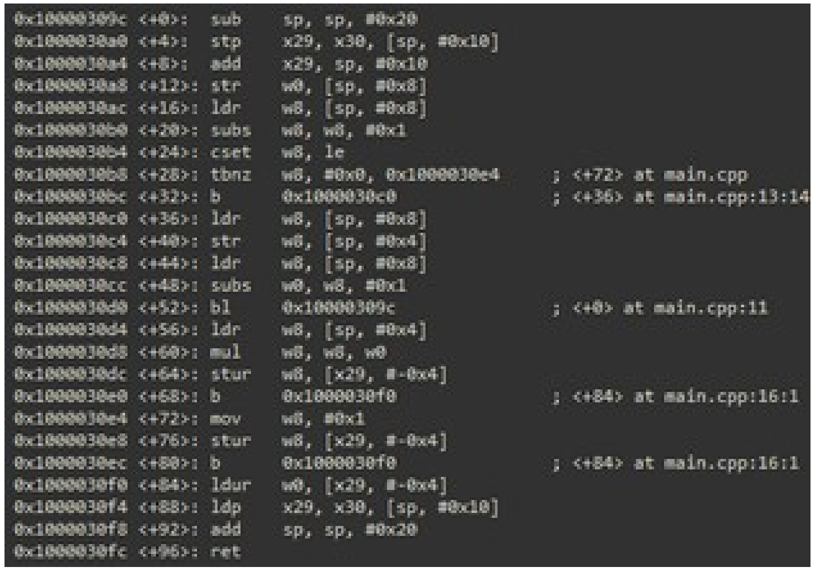
Identical semantics mapped different Instruction Set Architectures (ISA)

Translation requires sound knowledge on the target ISA



```
00007FF79EF52320 89 4C 24 08          mov         dword ptr [rsp+8],ecx
00007FF79EF52324 55                   push        rbp
00007FF79EF52325 57                   push        rdi
00007FF79EF52326 48 81 EC E8 00 00 00 sub         rsp,0E8h
00007FF79EF5232D 48 8D 6C 24 20       lea         rbp,[rsp+20h]
00007FF79EF52332 48 8D 0D F0 0C 01 00 lea         rcx,[__F791F1DE_Lecture01@cpp (07FF79EF63029h)]
00007FF79EF52339 E8 AA F0 FF FF       call        __CheckForDebuggerJustMyCode (07FF79EF513E8h)
   if (p>1)
00007FF79EF5233E 83 BD E0 00 00 00 01 cmp         dword ptr [p],1
00007FF79EF52345 7E 1C                jle         f+43h (07FF79EF52363h)
      return p*f(p-1);
00007FF79EF52347 8B 85 E0 00 00 00    mov         eax,dword ptr [p]
00007FF79EF5234D FF C8                dec         eax
00007FF79EF5234F 8B C8                mov         ecx,eax
00007FF79EF52351 E8 75 EF FF FF       call        f (07FF79EF512C8h)
00007FF79EF52356 8B 8D E0 00 00 00    mov         ecx,dword ptr [p]
00007FF79EF5235C 0F AF C8             imul        ecx,eax
00007FF79EF5235F 8B C1                mov         eax,ecx
00007FF79EF52361 EB 05                jmp         f+48h (07FF79EF52368h)

   return 1;
00007FF79EF52363 B8 01 00 00 00       mov         eax,1
}
00007FF79EF52368 48 8D A5 C8 00 00 00 lea         rsp,[rbp+0C8h]
00007FF79EF5236F 5F                   pop         rdi
00007FF79EF52370 5D                   pop         rbp
00007FF79EF52371 C3                   ret
```



```
0x10000309c <+0>:  sub    sp, sp, #0x20
0x1000030a0 <+4>:  stp    x29, x30, [sp, #0x10]
0x1000030a4 <+8>:  add    x29, sp, #0x10
0x1000030a8 <+12>: str    w0, [sp, #0x8]
0x1000030ac <+16>: ldr    w8, [sp, #0x8]
0x1000030b0 <+20>: subs   w8, w8, #0x1
0x1000030b4 <+24>: cset   w8, le
0x1000030b8 <+28>: tbnz   w8, #0x0, 0x1000030e4      ; <+72> at main.cpp
0x1000030bc <+32>: b      0x1000030c0                ; <+36> at main.cpp:13:14
0x1000030c0 <+36>: ldr    w8, [sp, #0x8]
0x1000030c4 <+40>: str    w8, [sp, #0x4]
0x1000030c8 <+44>: ldr    w8, [sp, #0x8]
0x1000030cc <+48>: subs   w0, w8, #0x1
0x1000030d0 <+52>: bl     0x10000309c                ; <+0> at main.cpp:11
0x1000030d4 <+56>: ldr    w8, [sp, #0x4]
0x1000030d8 <+60>: mul    w8, w8, w0
0x1000030dc <+64>: stur   w8, [x29, #-0x4]
0x1000030e0 <+68>: b      0x1000030f0                ; <+84> at main.cpp:16:1
0x1000030e4 <+72>: mov    w8, #0x1
0x1000030e8 <+76>: stur   w8, [x29, #-0x4]
0x1000030ec <+80>: b      0x1000030f0                ; <+84> at main.cpp:16:1
0x1000030f0 <+84>: ldur   w0, [x29, #-0x4]
0x1000030f4 <+88>: ldp    x29, x30, [sp, #0x10]
0x1000030f8 <+92>: add    sp, sp, #0x20
0x1000030fc <+96>: ret
```

x64 - CISC

ArmV8 - RISC

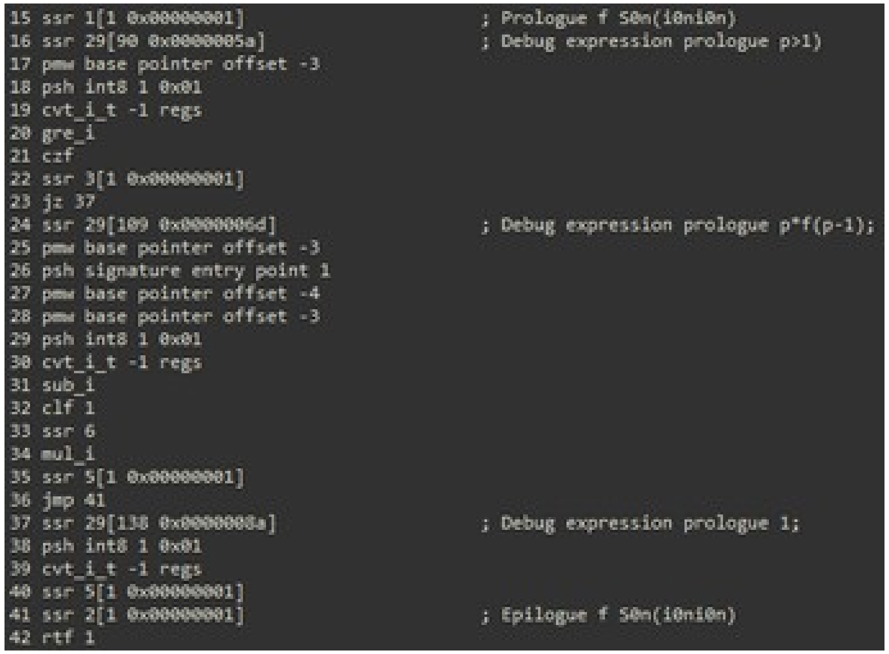
Architecture

```
int f(int p)
{
   if (p>1)
      return p*f(p-1);

   return 1;
}
```

Identical semantics mapped different Instruction Set Architectures (ISA)

ISA can also be defined at software level that may be interpreted or translated further targeting another lower level ISA.



```
15 ssr 1[1 0x00000001]                         ; Prologue f S0n(i0ni0n)
16 ssr 29[90 0x0000005a]                       ; Debug expression prologue p>1)
17 pmw base pointer offset -3
18 psh int8 1 0x01
19 cvt_i_t -1 regs
20 gre_i
21 czf
22 ssr 3[1 0x00000001]
23 jz 37
24 ssr 29[109 0x0000006d]                      ; Debug expression prologue p*f(p-1);
25 pmw base pointer offset -3
26 psh signature entry point 1
27 pmw base pointer offset -4
28 pmw base pointer offset -3
29 psh int8 1 0x01
30 cvt_i_t -1 regs
31 sub_i
32 clf 1
33 ssr 6
34 mul_i
35 ssr 5[1 0x00000001]
36 jmp 41
37 ssr 29[138 0x0000008a]                      ; Debug expression prologue 1;
38 psh int8 1 0x01
39 cvt_i_t -1 regs
40 ssr 5[1 0x00000001]
41 ssr 2[1 0x00000001]                         ; Epilogue f S0n(i0ni0n)
42 rtf 1
```

A sample stack machine

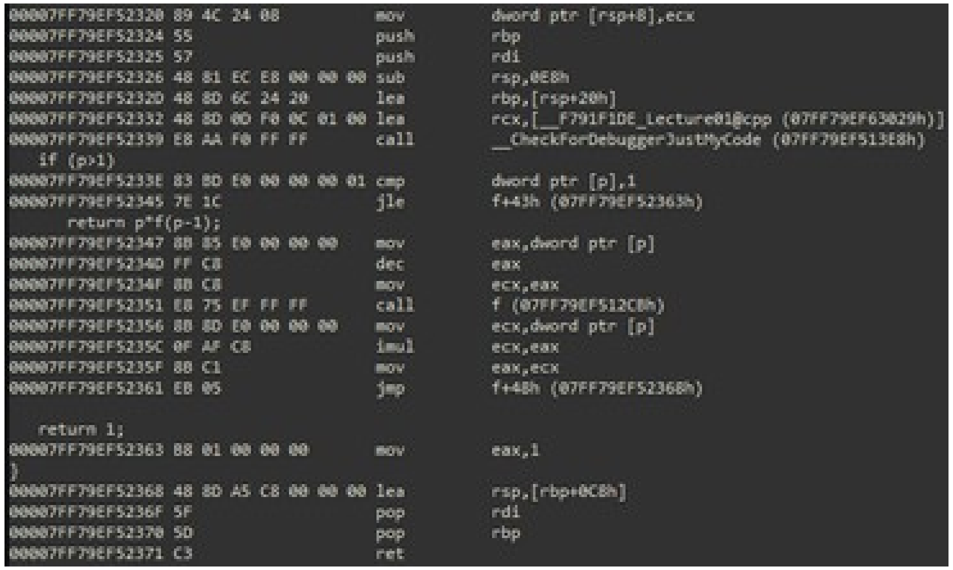
Architecture

```
int f(int p)
{
   if (p>1)
      return p*f(p-1);

   return 1;
}
```

Identical semantics mapped different Instruction Set Architectures (ISA)

Application Binary Interface (ABI) as OS dependent architectural manifestation.

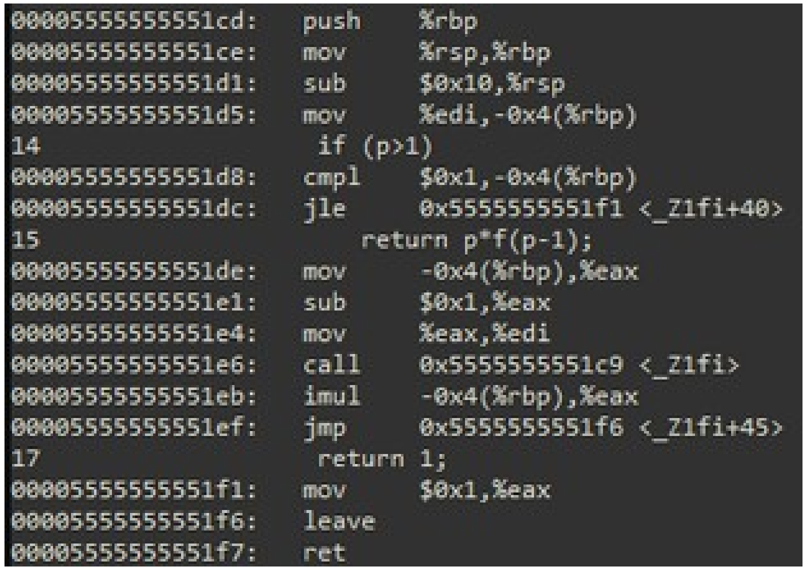


```
00007FF79EF52320 89 4C 24 08          mov         dword ptr [rsp+8],ecx
00007FF79EF52324 55                   push        rbp
00007FF79EF52325 57                   push        rdi
00007FF79EF52326 48 81 EC E8 00 00 00 sub         rsp,0E8h
00007FF79EF5232D 48 8D 6C 24 20       lea         rbp,[rsp+20h]
00007FF79EF52332 48 8D 0D F0 0C 01 00 lea         rcx,[__F791F1DE_Lecture01@cpp (07FF79EF63029h)]
00007FF79EF52339 E8 AA F0 FF FF       call        __CheckForDebuggerJustMyCode (07FF79EF513E8h)
   if (p>1)
00007FF79EF5233E 83 BD E0 00 00 00 01 cmp         dword ptr [p],1
00007FF79EF52345 7E 1C                jle         f+43h (07FF79EF52363h)
      return p*f(p-1);
00007FF79EF52347 8B 85 E0 00 00 00    mov         eax,dword ptr [p]
00007FF79EF5234D FF C8                dec         eax
00007FF79EF5234F 8B C8                mov         ecx,eax
00007FF79EF52351 E8 75 EF FF FF       call        f (07FF79EF512C8h)
00007FF79EF52356 8B 8D E0 00 00 00    mov         ecx,dword ptr [p]
00007FF79EF5235C 0F AF C8             imul        ecx,eax
00007FF79EF5235F 8B C1                mov         eax,ecx
00007FF79EF52361 EB 05                jmp         f+48h (07FF79EF52368h)

   return 1;
00007FF79EF52363 B8 01 00 00 00       mov         eax,1
}
00007FF79EF52368 48 8D A5 C8 00 00 00 lea         rsp,[rbp+0C8h]
00007FF79EF5236F 5F                   pop         rdi
00007FF79EF52370 5D                   pop         rbp
00007FF79EF52371 C3                   ret
```

x64 – Windows ABI

https://learn.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-170

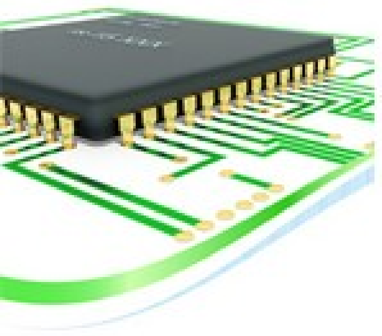


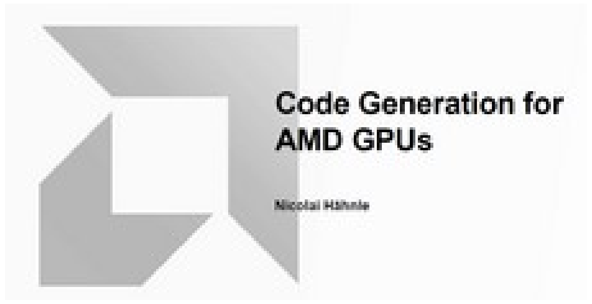
```
00005555555551cd:   push    %rbp
00005555555551ce:   mov     %rsp,%rbp
00005555555551d1:   sub     $0x10,%rsp
00005555555551d5:   mov     %edi,-0x4(%rbp)
14                   if (p>1)
00005555555551d8:   cmpl    $0x1,-0x4(%rbp)
00005555555551dc:   jle     0x5555555551f1 <_Z1fi+40>
15                      return p*f(p-1);
00005555555551de:   mov     -0x4(%rbp),%eax
00005555555551e1:   sub     $0x1,%eax
00005555555551e4:   mov     %eax,%edi
00005555555551e6:   call    0x5555555551c9 <_Z1fi>
00005555555551eb:   imul    -0x4(%rbp),%eax
00005555555551ef:   jmp     0x5555555551f6 <_Z1fi+45>
17                    return 1;
00005555555551f1:   mov     $0x1,%eax
00005555555551f6:   leave
00005555555551f7:   ret
```

x64 – Linux ABI

https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/linux-x64-calling-convention-stack-frame

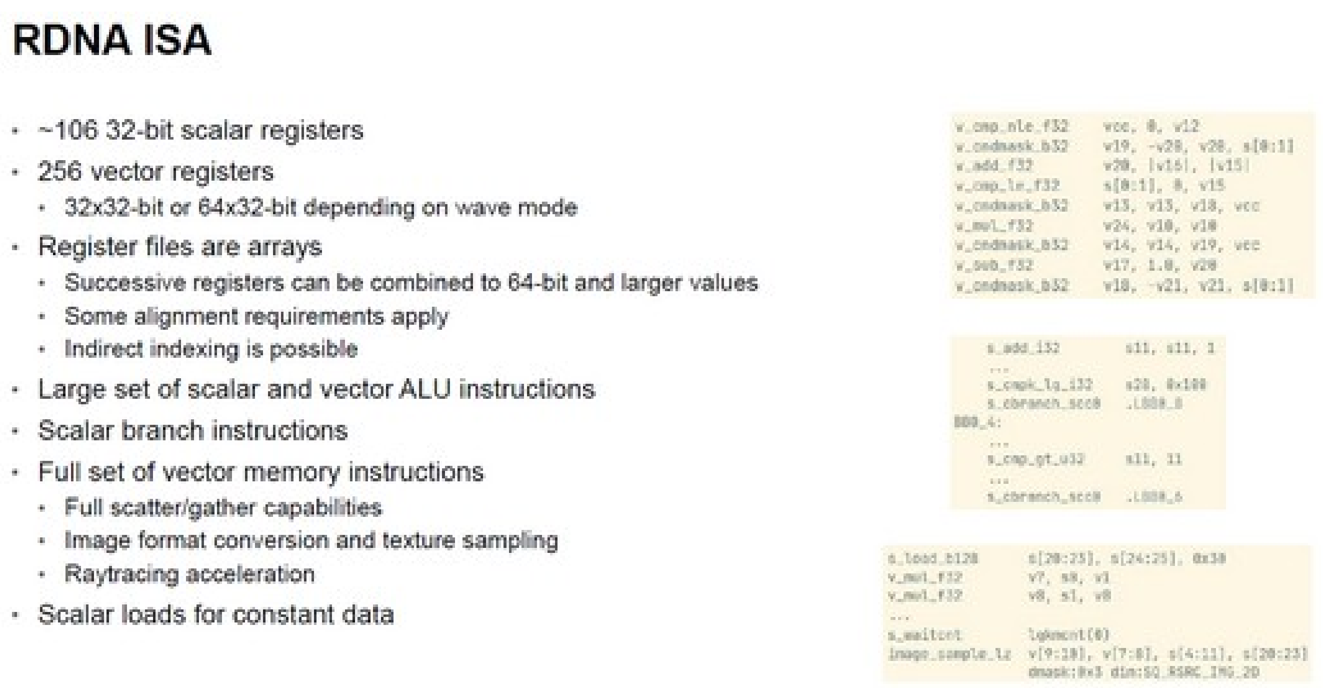
Architecture





Code Generation for AMD GPUs

Nicolai Hähnle



RDNA ISA

- ~106 32-bit scalar registers
- 256 vector registers
  - 32x32-bit or 64x32-bit depending on wave mode
- Register files are arrays
  - Successive registers can be combined to 64-bit and larger values
  - Some alignment requirements apply
  - Indirect indexing is possible
- Large set of scalar and vector ALU instructions
- Scalar branch instructions
- Full set of vector memory instructions
  - Full scatter/gather capabilities
  - Image format conversion and texture sampling
  - Raytracing acceleration
- Scalar loads for constant data

```
v_cmp_nle_f32     vcc, 0, v12
v_cndmask_b32     v19, -v20, v20, s[0:1]
v_add_f32         v20, |v16|, |v15|
v_cmp_le_f32      s[0:1], 0, v15
v_cndmask_b32     v13, v13, v18, vcc
v_mul_f32         v24, v10, v10
v_cndmask_b32     v14, v14, v19, vcc
v_sub_f32         v17, 1.0, v20
v_cndmask_b32     v18, -v21, v21, s[0:1]
```

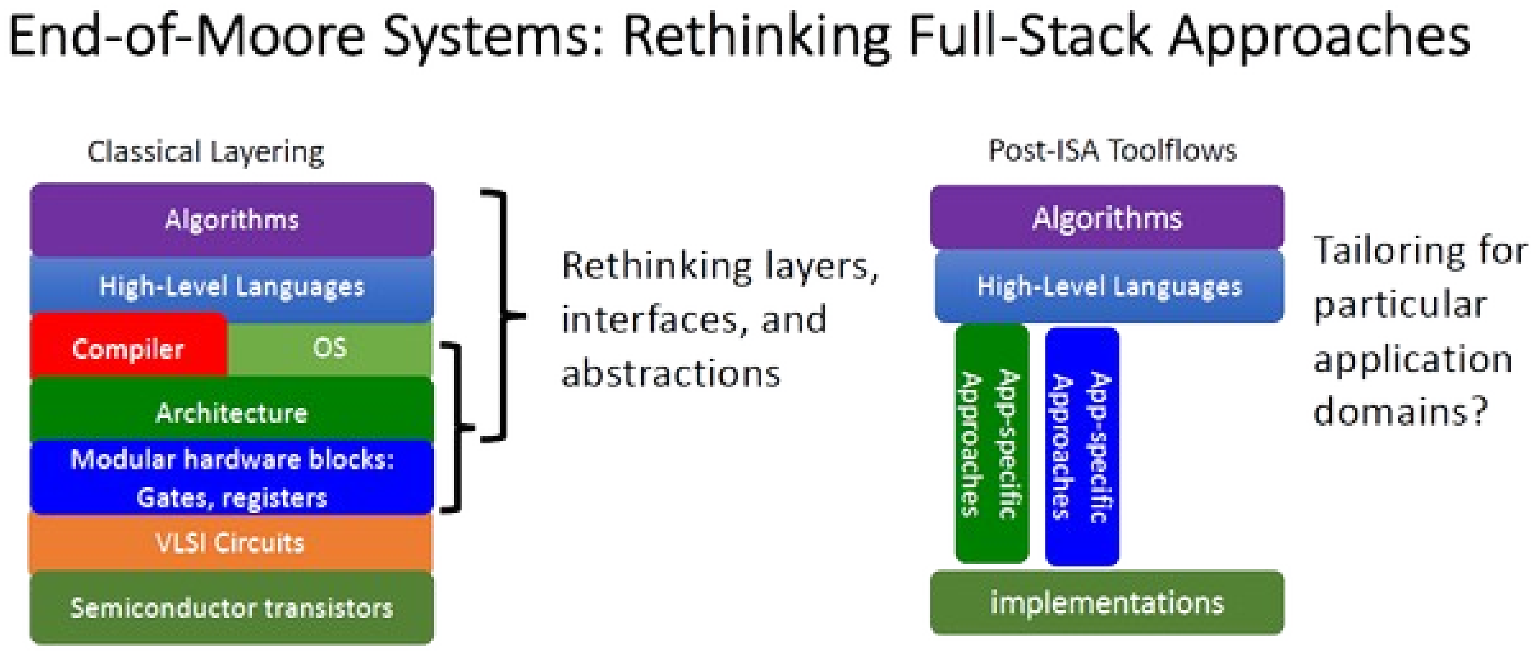
```
    s_add_i32         s11, s11, 1
    ...
    s_cmpk_lg_i32     s20, 0x100
    s_cbranch_scc0    .LBB0_8
BB0_4:
    ...
    s_cmp_gt_u32      s11, 11
    ...
    s_cbranch_scc0    .LBB0_6
```

```
s_load_b128        s[20:23], s[24:25], 0x30
v_mul_f32          v7, s8, v1
v_mul_f32          v8, s1, v8
...
s_waitcnt          lgkmcnt(0)
image_sample_lz    v[9:10], v[7:8], s[4:11], s[20:23]
                   dmask:0x3 dim:SQ_RSRC_IMG_2D
```

https://db.in.tum.de/teaching/ws2223/codegen/CodegenForGPUs.pdf?lang=en

Architecture

Application specific architectures have deep impact


End-of-Moore Systems: Rethinking Full-Stack Approaches
Classical Layering
Algorithms
High-Level Languages
Compiler
OS
Architecture
Modular hardware blocks: Gates, registers
VLSI Circuits
Semiconductor transistors
Rethinking layers, interfaces, and abstractions
Post-ISA Toolflows
Algorithms
High-Level Languages
App-specific Approaches
App-specific Approaches
implementations
Tailoring for particular application domains?

Quick View 1 / 4

Lexical Analysis

- Reserved words, tokens
- Regular expressions
- NFA, DFA structures
- Optimized structures, strategies
- Generators and recognizers
- Experiment

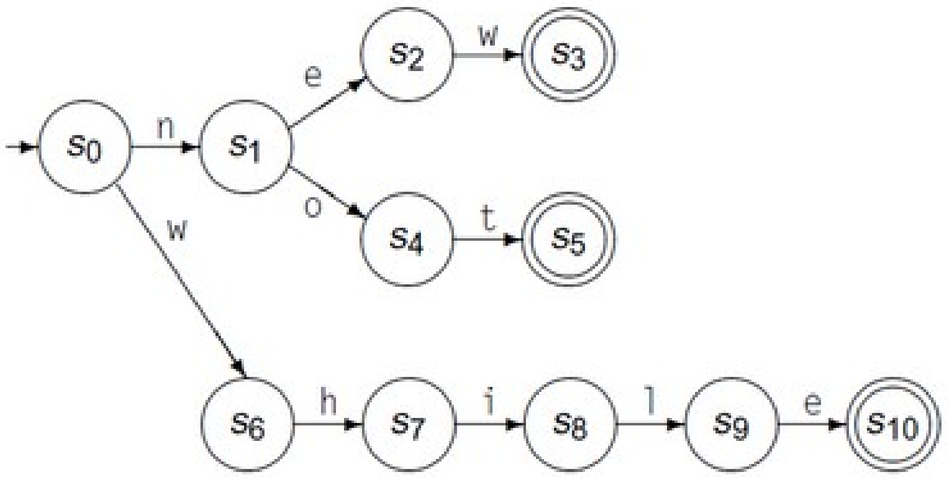


Diagram from “Cooper, K.D., Torczon, L.; Engineering A Compiler”

Quick View 2 / 4

Syntactical Analysis

- Context Free Grammars, BNF
- Derivations, sentential forms
- Parse trees, AST
- Parsing strategies
- Parsing algorithms
- Experiment

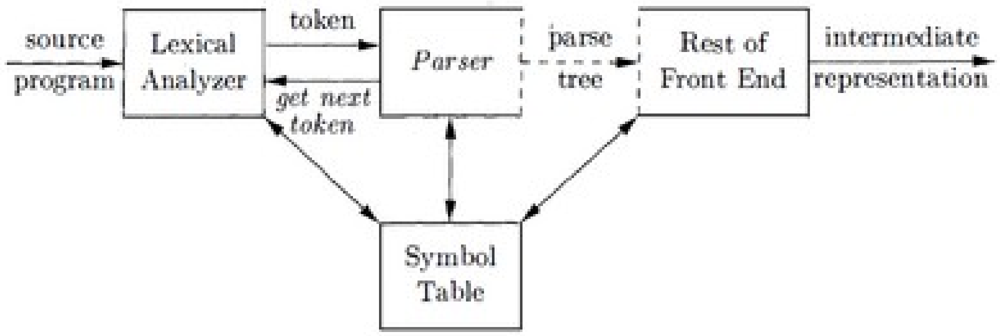


Diagram from “Aho, A.V, Ullman J.D, Sethi R., Lam M.S; Dragon Book”

Quick View 3 / 4

Semantic Analysis

- Meaning and typical patterns
- Type systems
- Attributes, SDD
- Representing “the meaning”
- Experiment

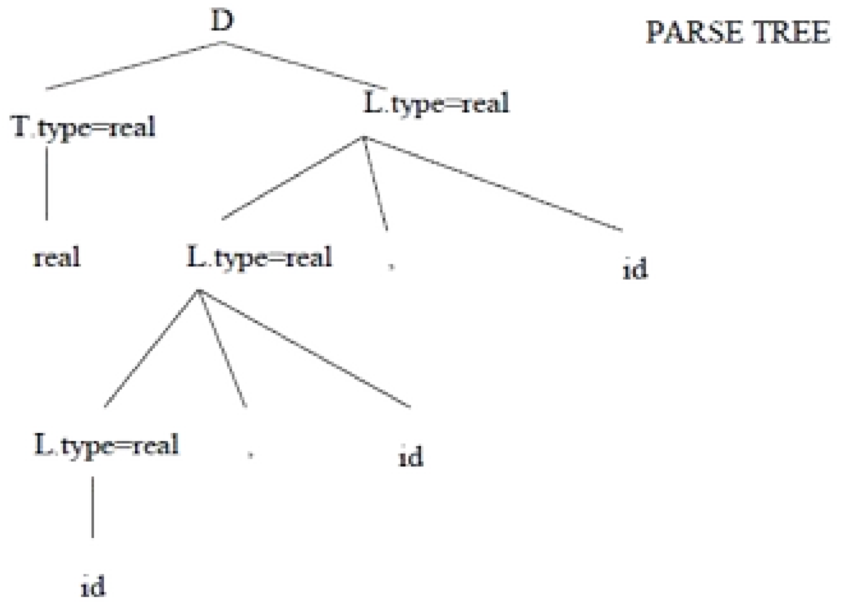
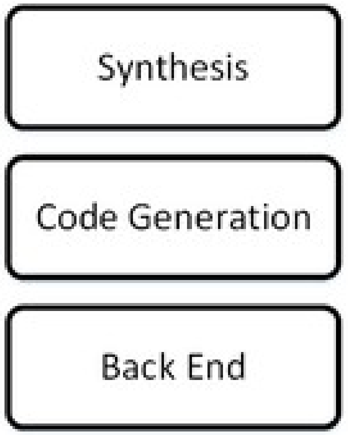


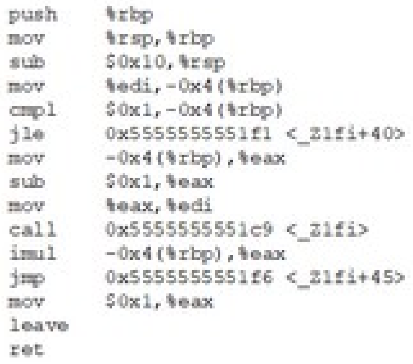
Diagram from CENG 444 course notes, Prof. Dr. Cem Bozşahin

Quick View 4 / 4

Towards Back End Processing

- Target architectures
- ABIs, calling conventions
- Intermediate code
- Optimizations
- Register allocation
- Compiler runtime
- Finally, writing target!
- Experiment





```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
mov     %edi,-0x4(%rbp)
cmpl    $0x1,-0x4(%rbp)
jle     0x5555555551f1 <_Z1fi+40>
mov     -0x4(%rbp),%eax
sub     $0x1,%eax
mov     %eax,%edi
call    0x5555555551c9 <_Z1fi>
imul    -0x4(%rbp),%eax
jmp     0x5555555551f6 <_Z1fi+45>
mov     $0x1,%eax
leave
ret
```

… and a few words on

- Data flow analysis
- Static single assignment forms
- Instruction scheduling