More Target Independent LLVM Bitcode
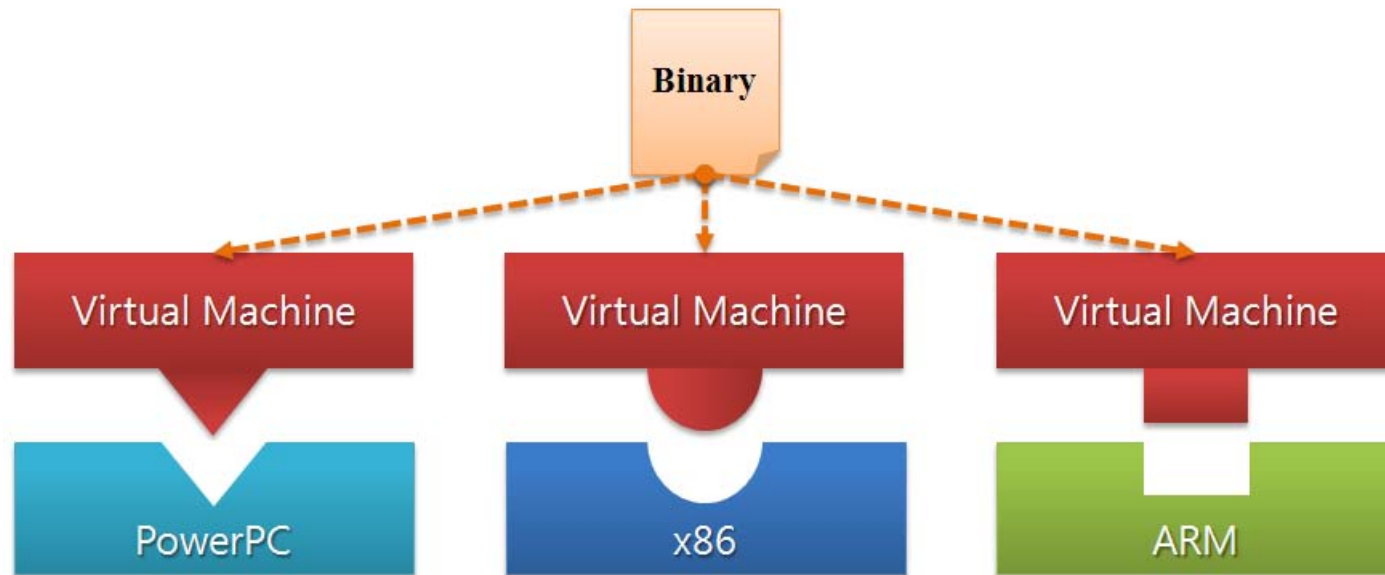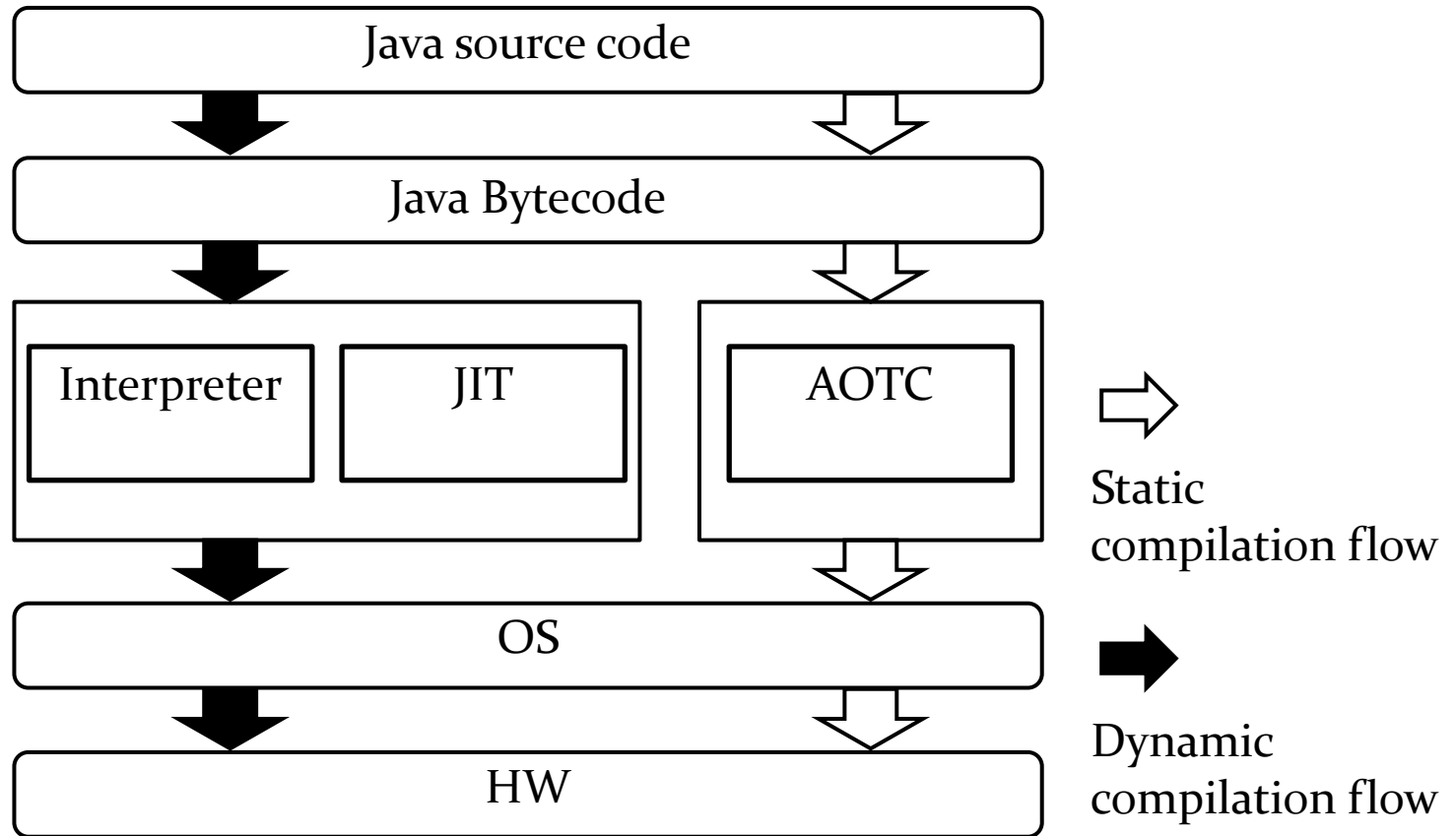
IMRC in KIST

Jin-Gu Kang

Contents

- Motivation
- Target Dependent Properties
- New Compilation Strategy
- More Target Independent Bitcode
- Application

Motivation

- A compiled code can be executed identically on all of machines using virtual machine.

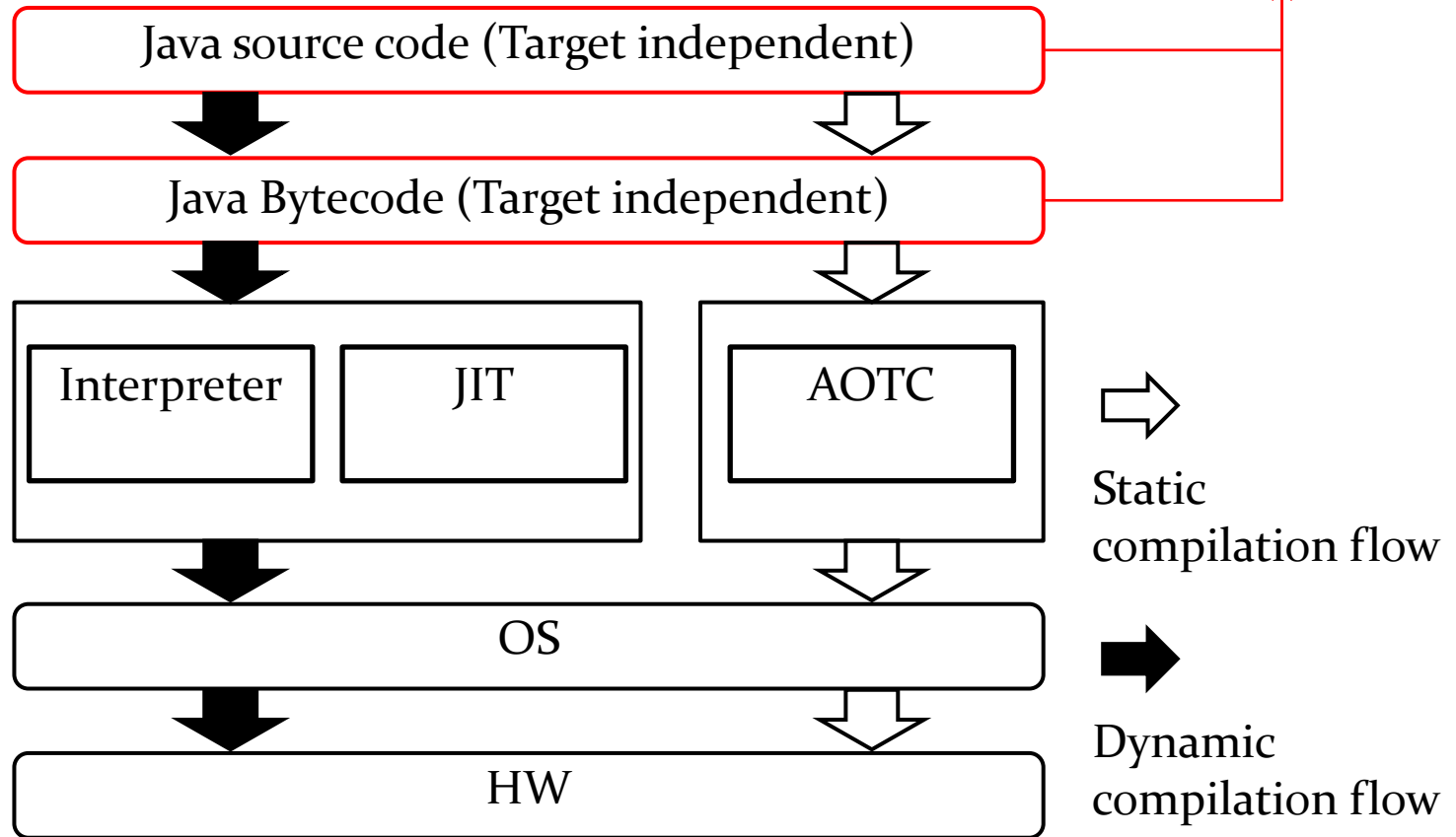- Is it possibe to use LLVM Bitcode like Java Bytecode??

Java Compilation Flow

Java Compilation Flow

Java language is not affetcted by specific machine.
Java bytecode is same among machines.

| Java source code (Target independent) |

| Java Bytecode (Target independent) |

| Interpreter | JIT | | AOTC |

Static compilation flow

| OS |

Dynamic compilation flow

| HW |

LLVM Compilation Flow
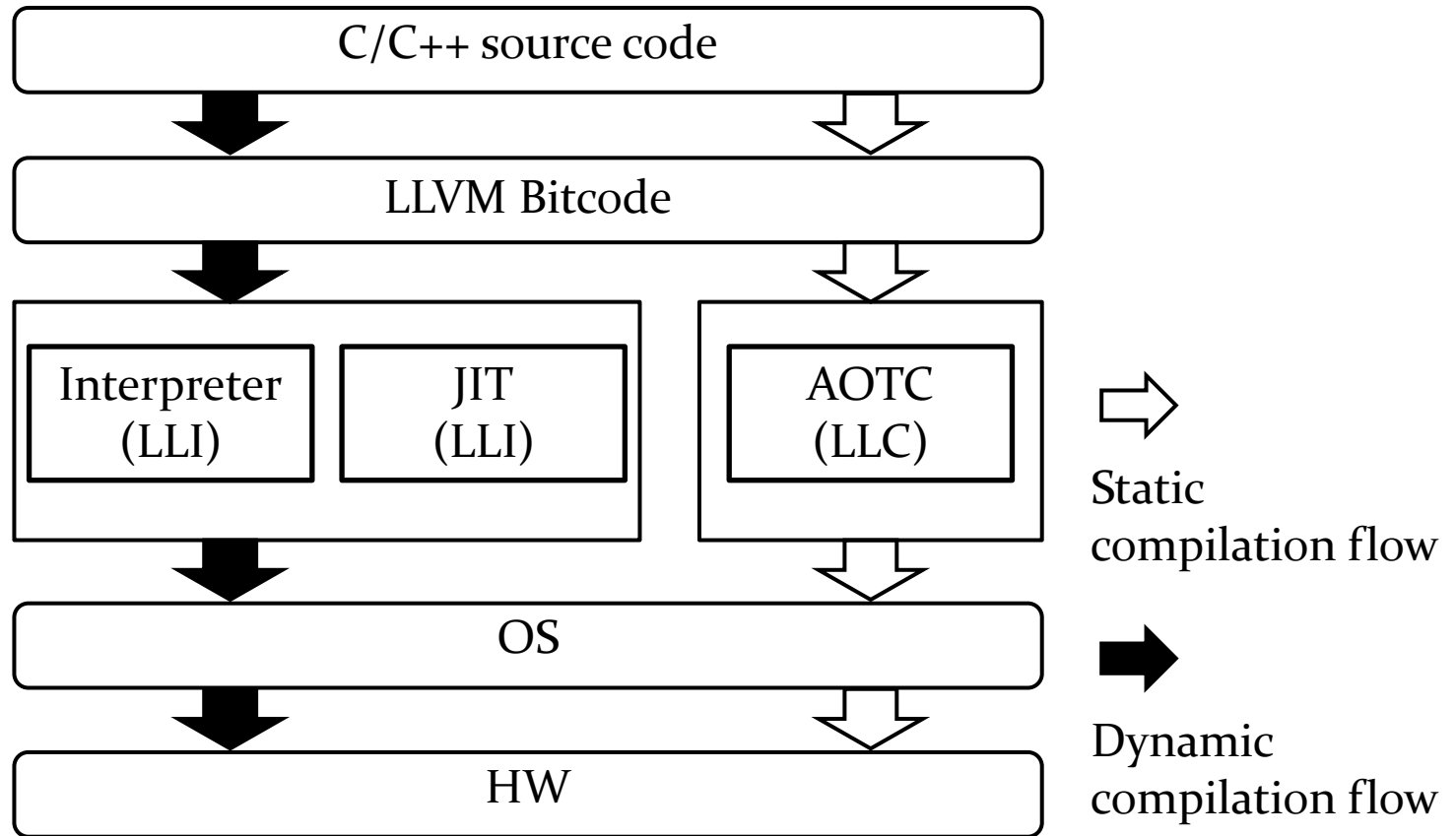
LLVM Compilation Flow

C/C++ language is affetcted by specific machine.
LLVM bitcode is not same among machines.
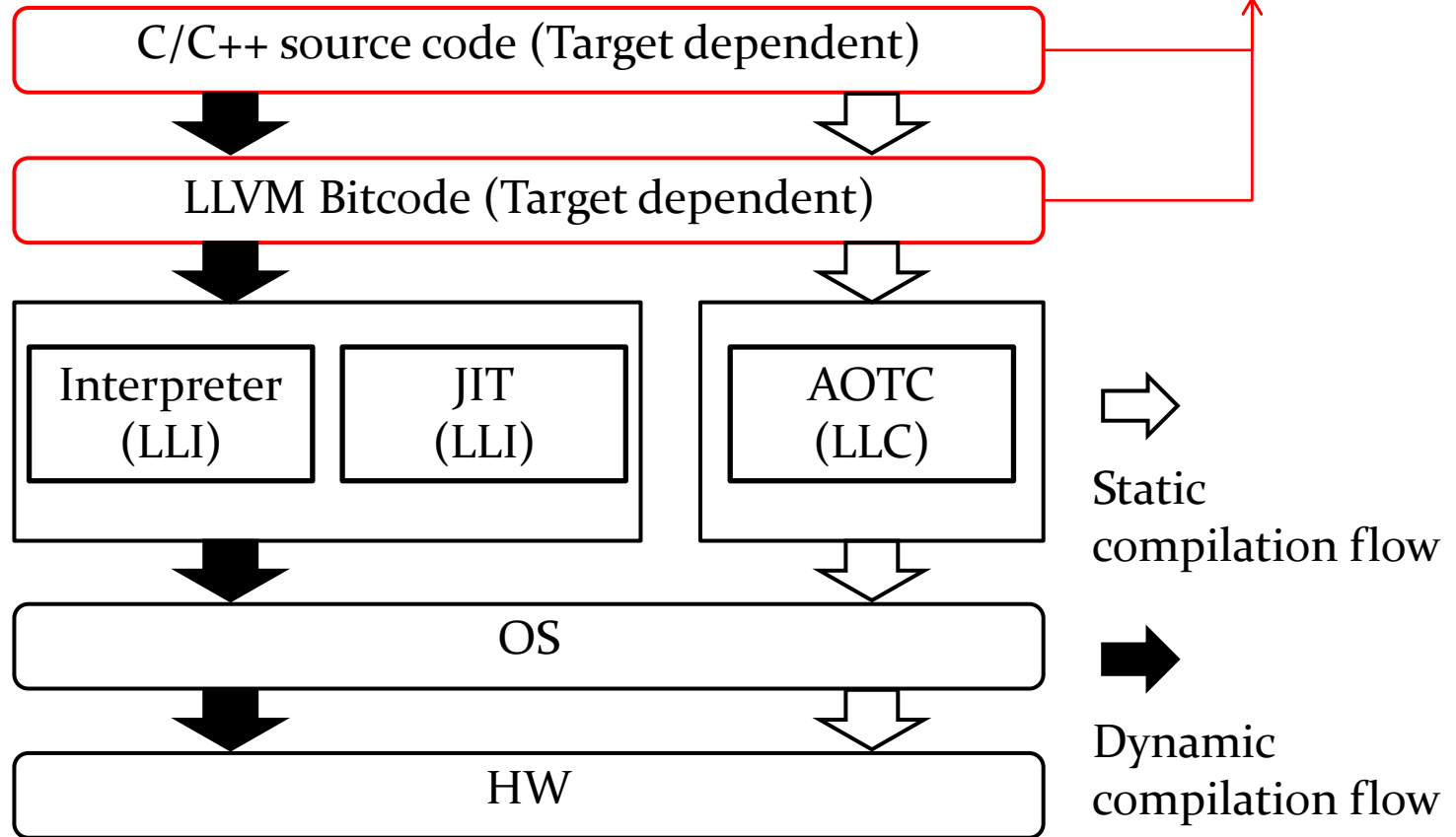
C/C++ source code (Target dependent)

LLVM Bitcode (Target dependent)

Interpreter (LLI) | JIT (LLI) | AOTC (LLC)

OS

HW

⇨ Static compilation flow

➡ Dynamic compilation flow

Target Dependent Properties

- LLVM frontend
  - LLVM frontend generates some target dependent bitcodes with common LLVM IR.
    - Function type
    - Sizeof() keyword
- LLVM IR
  - Bitcode does not support common IR in some cases.
    - Long double type
    - Struct type with bitfield
- Source language
  - Inline assembly
  - Target dependent source codes

Target Dependent Properties

- LLVM frontend
  - LLVM frontend generates some target dependent bitcodes with common LLVM IR.
    - Function type
    - Sizeof() keyword
- LLVM IR
  - Bitcode does not support common IR in some cases.
    - Long double type
    - Struct type with bitfield
- Source language
  - Inline assembly
  - Target dependent source codes

Function Type

- Function Type
  - ABI compatibility

```
1 struct operands {
2   int a;
3   char b;
4 };
5
6 int add(struct operands ops) {
7   return ops.a + ops.b;
8 }
9
10 int main(void) {
11   int c;
12   struct operands ops;
13
14   ops.a = 1;
15   ops.b = 2;
16   c = add(ops);
17
18   return 0;
19 }
```

Function Type

x86

```
28 define i32 @main() nounwind {
29 entry:
30   %retval = alloca i32                                 ; <i32*> [#uses=2]
31   %ops = alloca %struct.operands                       ; <%struct.operands*> [#uses=3]
32   %c = alloca i32                                      ; <i32*> [#uses=1]
33   %0 = alloca i32                                      ; <i32*> [#uses=2]
34   %"alloca point" = bitcast i32 0 to i32               ; <i32> [#uses=0]
35   %1 = getelementptr inbounds %struct.operands* %ops, i32 0, i32 0 ; <i32*> [#uses=1]
36   store i32 1, i32* %1, align 4
37   %2 = getelementptr inbounds %struct.operands* %ops, i32 0, i32 1 ; <i8*> [#uses=1]
38   store i8 2, i8* %2, align 4
39   %3 = call i32 @add(%struct.operands* byval align 4 %ops) nounwind ; <i32> [#uses=1]
```

ARM

```
36 define arm_aapcscc i32 @main() nounwind {
37 entry:
38   %retval = alloca i32                                 ; <i32*> [#uses=2]
39   %ops = alloca %struct.operands                       ; <%struct.operands*> [#uses=3]
40   %c = alloca i32                                      ; <i32*> [#uses=1]
41   %0 = alloca i32                                      ; <i32*> [#uses=2]
42   %"alloca point" = bitcast i32 0 to i32               ; <i32> [#uses=0]
43   %1 = getelementptr inbounds %struct.operands* %ops, i32 0, i32 0 ; <i32*> [#uses=1]
44   store i32 1, i32* %1, align 4
45   %2 = getelementptr inbounds %struct.operands* %ops, i32 0, i32 1 ; <i8*> [#uses=1]
46   store i8 2, i8* %2, align 4
47   %3 = bitcast %struct.operands* %ops to %0*           ; <%0*> [#uses=1]
48   %elt = getelementptr inbounds %0* %3, i32 0, i32 0 ; <[2 x i32]*> [#uses=2]
49   %elt1 = getelementptr inbounds [2 x i32]* %elt, i32 0, i32 0 ; <i32*> [#uses=1]
50   %val = load i32* %elt1                               ; <i32> [#uses=1]
51   %elt2 = getelementptr inbounds [2 x i32]* %elt, i32 0, i32 1 ; <i32*> [#uses=1]
52   %val3 = load i32* %elt2                              ; <i32> [#uses=1]
53   %4 = call arm_aapcscc  i32 @add(i32 %val, i32 %val3) nounwind ; <i32> [#uses=1]
```

Sizeof() keyword

```
1 int main(void) {
2    int a = sizeof(long double);
3
4    return 0;
5 }
```

```
3  target triple = "i386-pc-linux-gnu"
4
5  define i32 @main() nounwind {
6  entry:
7    %retval = alloca i32
8    %a = alloca i32
9    %0 = alloca i32
10   %"alloca point" = bitcast i32 0 to i32
11   store i32 12, i32* %a, align 4
12   store i32 0, i32* %0, align 4
13   %1 = load i32* %0, align 4
14   store i32 %1, i32* %retval, align 4
15   br label %return
```

ARM

```
3  target triple = "armv5-none-linux-gnueabi"
4
5  define arm_aapcscc i32 @main() nounwind {
6  entry:
7    %retval = alloca i32
8    %a = alloca i32
9    %0 = alloca i32
10   %"alloca point" = bitcast i32 0 to i32
11   store i32 8, i32* %a, align 4
12   store i32 0, i32* %0, align 4
13   %1 = load i32* %0, align 4
14   store i32 %1, i32* %retval, align 4
15   br label %return
```

Target Dependent Properties

- LLVM frontend
  - LLVM frontend generates some target dependent bitcodes with common LLVM IR.
    - Function type
    - Sizeof() keyword
- LLVM IR
  - LLVM does not support common IR in some cases.
    - Long double type
    - Struct type with bitfield
- Source language
  - Inline assembly
  - Target dependent source codes

KIST
Korea Institute of
Science and Technology

Target Dependent Properties

- long double type
  - Various encoding formats

IEEE double precision format

ARM

IEEE extended precision format

X86

IEEE quadruple precision format

PPC
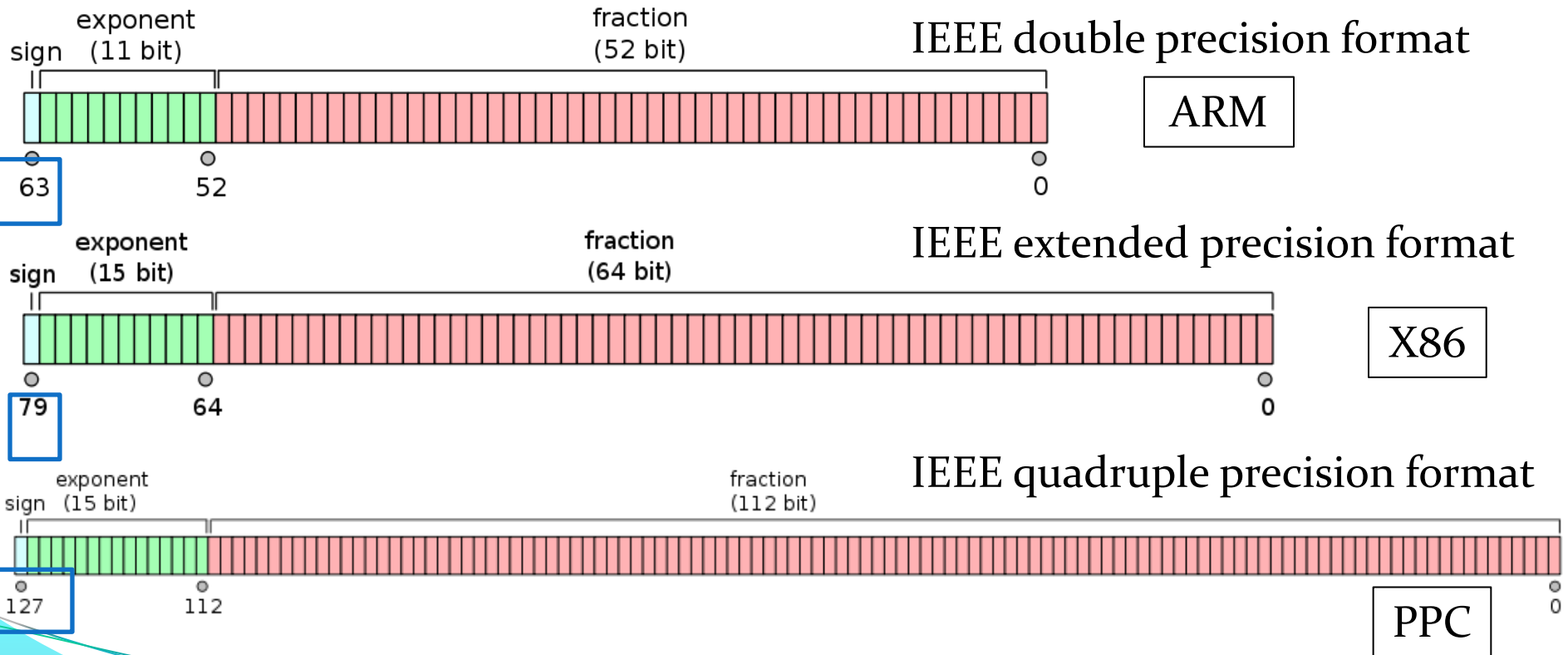
Target Dependent Properties

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5      long double a = 3.14;
 6      long double b = 2.2;
 7      long double c;
 8
 9      c = a + b;
10
11      printf("c=%Lf\n", c);
12
13      return 0;
14  }
```

LLVM does not support common IR type for long double type.

```
%"alloca_point" = bitcast i32 0 to i32
store x86_fp80 0xK4000C8F5C28F5C28F800, x86_fp80* %a, align 4
store x86_fp80 0xK40008CCCCCCCCCCCCD000, x86_fp80* %b, align 4
%1 = load x86_fp80* %a, align 4
%2 = load x86_fp80* %b, align 4
%3 = fadd x86_fp80 %1, %2                    ;
```

LLVM Bitcode For X86

```
%"alloca_point" = bitcast i32 0 to i32
store double 3.140000e+00, double* %a, align 8
store double 2.200000e+00, double* %b, align 8
%1 = load double* %a, align 8
%2 = load double* %b, align 8
%3 = fadd double %1, %2
```

LLVM Bitcode for ARM

Target Dependent Properties

Application

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      long double a = 3.14;
6      long double b = 2.2;
7      long double c;
8
9      c = a + b;
10
11     printf("c=%Lf\n", c);
12
13     return 0;
14 }
```

Source for ARM

IEEE_double format encoding variable "c" as parameter for printf function

sign | exponent (11 bit) | fraction (52 bit)

63   52   0
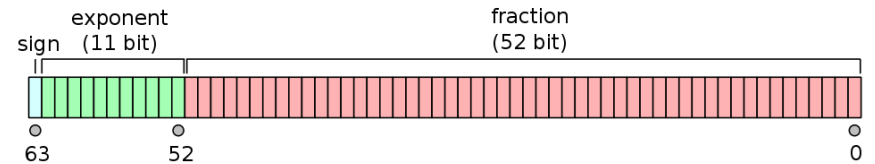
Library

System X86

Printf function waits X86_fp80 format encoding Variable for "%Lf" on X86 architecture
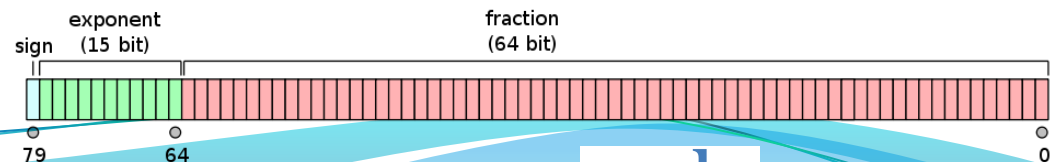
sign | exponent (15 bit) | fraction (64 bit)

79   64   0

Target Dependent Properties

- Strcut type with bitfield

ARM

a							
b							
c							
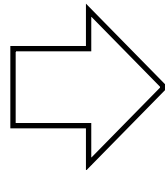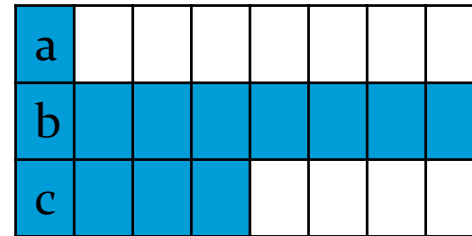
Memory

<{ i8, [7 x i8], i64, i32, [4 x i8]}>

LLVM IR

```
struct foo
{
    char a:4;
    long long b:61;
    int c:30;
};
```

x86

a				b			
				c			
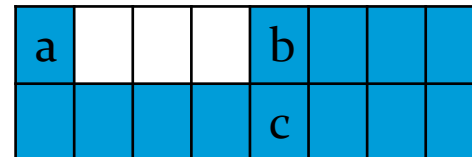
Memory

<{ i8, [3 x i8], i64, i32 }>

LLVM IR

Target Dependent Properties
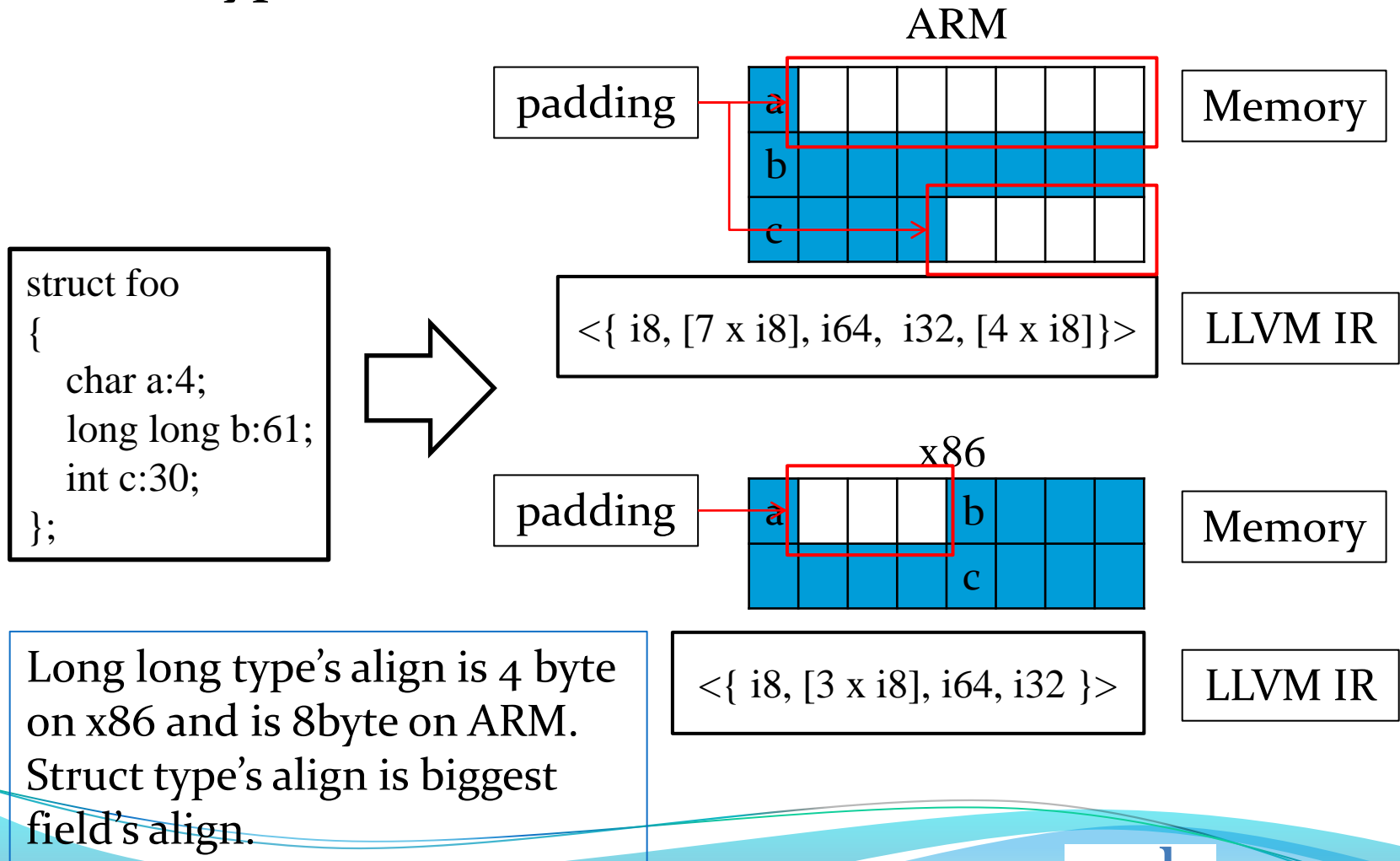
- Strcut type with bitfield

ARM

padding

a

b

c

Memory

```
struct foo
{
    char a:4;
    long long b:61;
    int c:30;
};
```

<{ i8, [7 x i8], i64, i32, [4 x i8]}>

LLVM IR

x86

padding

a

b

c

Memory

<{ i8, [3 x i8], i64, i32 }>

LLVM IR

Long long type's align is 4 byte on x86 and is 8byte on ARM. Struct type's align is biggest field's align.

Target Dependent Properties

- LLVM frontend
  - LLVM frontend generates some target dependent bitcodes with common LLVM IR.
    - Function type → ABI compatibility
    - Sizeof() keyword
- LLVM IR
  - Bitcode does not support common IR in some cases.
    - Long double type
    - Struct type with bitfield
- Source language
  - Inline assembly
  - Target dependent source codes

Discard

New Compilation Strategy

C/C++ source code

↓

Target Independent Bitcode (Wordcode)

↓

Translator

↓

Target Dependent Bitcode (Wordcode)

↓

AOTC
(LLC)

↓

OS

↓

HW

New Compilation Strategy

```
┌─────────────────────────────────────────────┐
│             C/C++ source code                │
└─────────────────────────────────────────────┘
                      ⬇
┌─────────────────────────────────────────────┐
│    Target Independent Bitcode (Wordcode)     │
└─────────────────────────────────────────────┘
                      ⬇
        ┌─────────────────────────────┐
        │  ┌───────────────────────┐  │
        │  │      Translator       │  │
        │  └───────────────────────┘  │
        └─────────────────────────────┘
                      ⬇
┌─────────────────────────────────────────────┐
│     Target Dependent Bitcode (Wordcode)      │
└─────────────────────────────────────────────┘
                      ⬇
        ┌─────────────────────────────┐
        │  ┌───────────────────────┐  │
        │  │         AOTC          │  │
        │  │         (LLC)         │  │
        │  └───────────────────────┘  │
        └─────────────────────────────┘
                      ⬇
┌─────────────────────────────────────────────┐
│                     OS                       │
└─────────────────────────────────────────────┘
                      ⬇
┌─────────────────────────────────────────────┐
│                     HW                       │
└─────────────────────────────────────────────┘
```
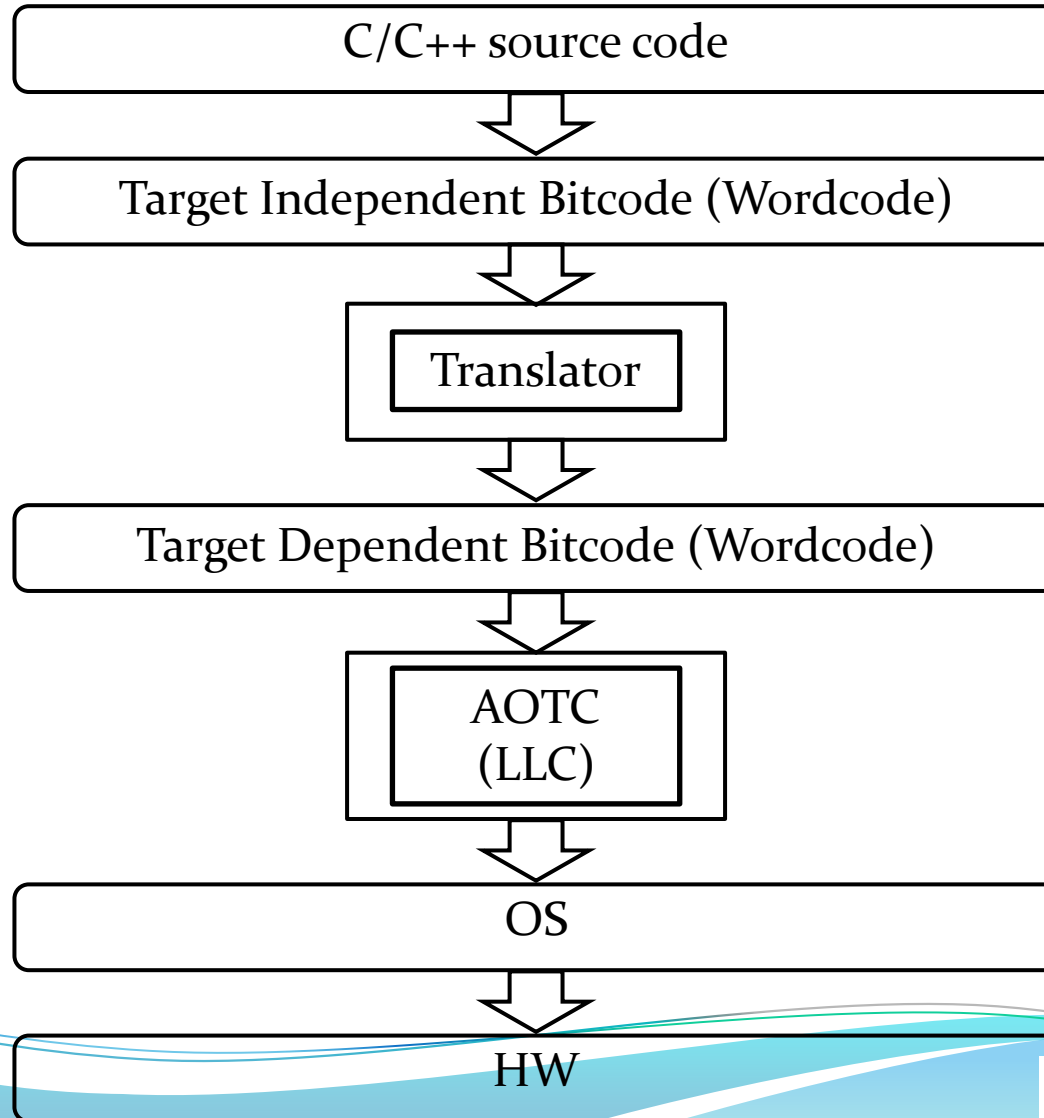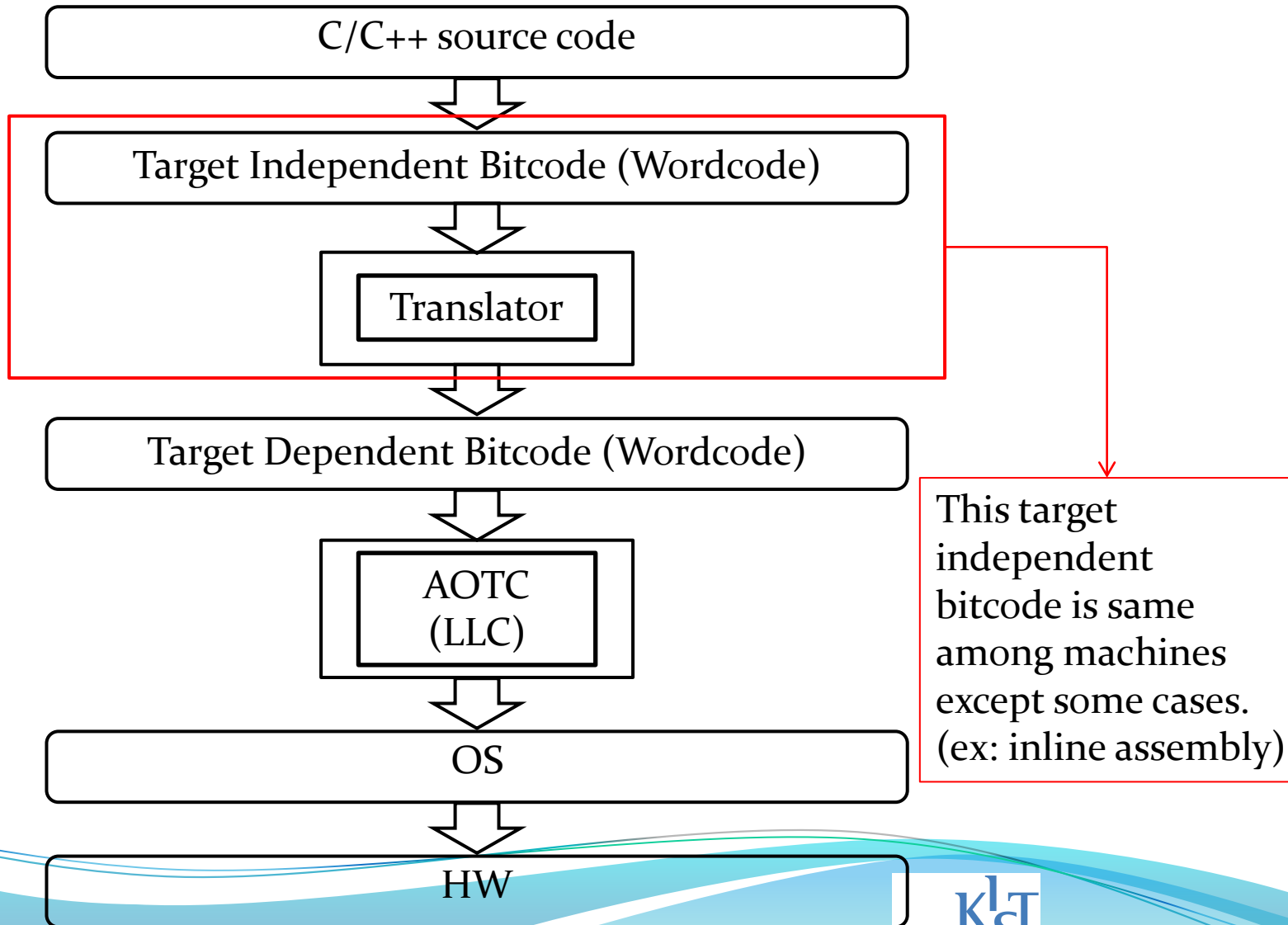
This target independent bitcode is same among machines except some cases. (ex: inline assembly)

New Compilation Strategy

```
┌─────────────────────────────────────────┐
│           C/C++ source code              │
└─────────────────────────────────────────┘
                    ▼
┌─────────────────────────────────────────┐
│   Target Independent Bitcode (Wordcode)  │
└─────────────────────────────────────────┘
                    ▼
┌─────────────────────────────────────────┐
│          ┌──────────────────┐           │
│          │    Translator    │           │
│          └──────────────────┘           │
│                    ▼                     │
│  ┌─────────────────────────────────────┐ │
│  │ Target Dependent Bitcode (Wordcode) │ │
│  └─────────────────────────────────────┘ │
└─────────────────────────────────────────┘
                    ▼
┌─────────────────────────────────────────┐
│          ┌──────────────────┐           │
│          │       AOTC       │           │
│          │       (LLC)      │           │
│          └──────────────────┘           │
└─────────────────────────────────────────┘
                    ▼
┌─────────────────────────────────────────┐
│                  OS                      │
└─────────────────────────────────────────┘
                    ▼
┌─────────────────────────────────────────┐
│                  HW                      │
└─────────────────────────────────────────┘
```
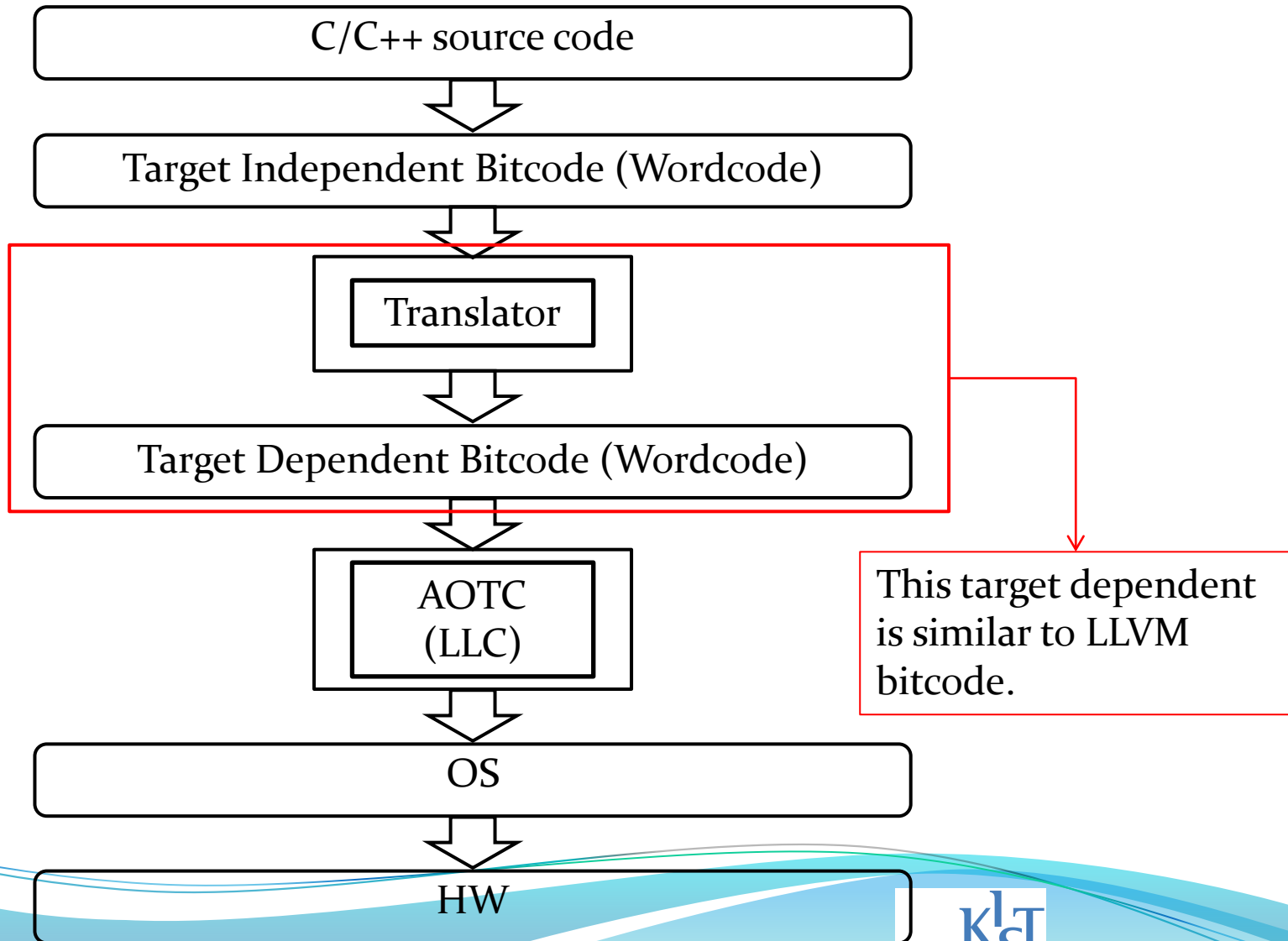
This target dependent is similar to LLVM bitcode.

Wordcode

- Wordcode can be target dependent or target independent.
  - Target independent Wordcode
    - IntegerTypes with align information from source code
    - StructType with bitfield
    - Long double type
    - Etc…
  - Target dependent Wordcode
    - It is similar to LLVM bitcode.
- Wordcode is applied to almost passes on LLVM.

Wordcode

Target independent Wordcode


```
1  int main (void) {
2    char a;
3    short b;
4    int c;
5    long d;
6    long long e;
7    float f;
8    double g;
9
10   return 0;
11 }
```

```
1  ; ModuleID = 'a.c'
2  target datalayout = "e-p:32:32:32-i1:8:8-i8:8:8-
   -v64:64:64-v128:128:128-a0:0:64-f192:32:32"
3  target triple = "ovm-none-linux"
4
5  define i32(int) @main() nounwind {
6  entry:
7    %retval = alloca i32(int)
8    %g = alloca double
9    %f = alloca float
10   %e = alloca i64(longlong)
11   %d = alloca i32(long)
12   %c = alloca i32(int)
13   %b = alloca i16(short)
14   %a = alloca i8(char)
15   %0 = alloca i32(int)
16   %"alloca point" = bitcast i32() 0 to i32()
17   store i32(int) 0, i32(int)* %0, align 4
18   %1 = load i32(int)* %0, align 4
19   store i32(int) %1, i32(int)* %retval, align 4
20   br label %return
21
22 return:
23   %retval1 = load i32(int)* %retval
24   ret i32(int) %retval1
25 }
```

Wordcode

Target independent Wordcode

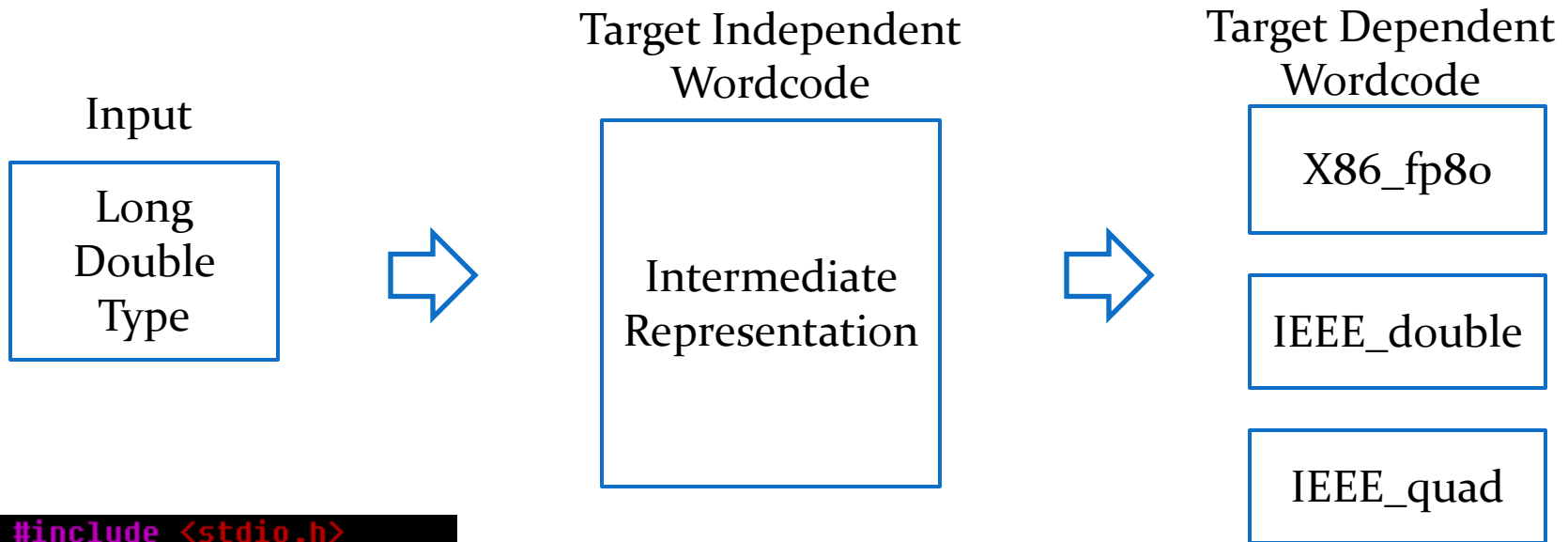Target dependent Wordcode (ARM)


```
1  ; ModuleID = 'a.c'
2  target datalayout = "e-p:32:32:32-i1:8:8:8-i8:8:8-
   -v64:64:64-v128:128:128-a0:0:64-f192:32:32"
3  target triple = "ovm-none-linux"
4
5  define i32(int) @main() nounwind {
6  entry:
7    %retval = alloca i32(int)
8    %g = alloca double
9    %f = alloca float
10   %e = alloca i64(longlong)
11   %d = alloca i32(long)
12   %c = alloca i32(int)
13   %b = alloca i16(short)
14   %a = alloca i8(char)
15   %0 = alloca i32(int)
16   %"alloca point" = bitcast i32() 0 to i32()
17   store i32(int) 0, i32(int)* %0, align 4
18   %1 = load i32(int)* %0, align 4
19   store i32(int) %1, i32(int)* %retval, align 4
20   br label %return
21
22 return:
23   %retval1 = load i32(int)* %retval
24   ret i32(int) %retval1
25 }
```

```
1  ; ModuleID = 'a.mod.opt.arm.bc'
2  target datalayout = "e-p:32:32:32-i1:8:8-i8:
   -v64:64:64-v128:128:128-a0:0:64"
3  target triple = "armv5-none-linux-gnueabi"
4
5  define i32() @main() nounwind {
6  entry:
7    %retval = alloca i32()
8    %g = alloca double
9    %f = alloca float
10   %e = alloca i64()
11   %d = alloca i32()
12   %c = alloca i32()
13   %b = alloca i16()
14   %a = alloca i8()
15   %0 = alloca i32()
16   %"alloca point" = bitcast i32() 0 to i32()
17   store i32() 0, i32()* %0, align 4
18   %1 = load i32()* %0, align 4
19   store i32() %1, i32()* %retval, align 4
20   br label %return
21
22 return:
23   %retval1 = load i32()* %retval
24   ret i32() %retval1
25 }
```

Long Double Type

Input

Target Independent
Wordcode

Target Dependent
Wordcode

Long Double Type		Intermediate Representation		X86_fp8o

IEEE_double

IEEE_quad

```c
1  #include <stdio.h>
2
3  int main(void) {
4      long double a = 3.14;
5      long double b = 2.2;
6      long double c;
7
8      c = a + b;
9
10     printf("c=%Lf\n", c);
11     return 0;
12 }
```

KIST
Korea Institute of
Science and Technology

Long Double Type

Target independent Wordcode

```
 7  define i32(int) @main() nounwind {
 8  entry:
 9    %retval = alloca i32(int)                         ; <i32(int)*> [#uses=2]
10    %c = alloca long_double                           ; <long_double*> [#uses=2]
11    %b = alloca long_double                           ; <long_double*> [#uses=2]
12    %a = alloca long_double                           ; <long_double*> [#uses=2]
13    %0 = alloca i32(int)                              ; <i32(int)*> [#uses=2]
14    %"alloca point" = bitcast i32() 0 to i32()        ; <i32()> [#uses=0]
15    store long_double 0xL00000000000000810000000000000000C8F5C28F5C28F800, long_double* %a, align 8
16    store long_double 0xL00000000000000810000000000000008CCCCCCCCCCCD000, long_double* %b, align 8
17    %1 = load long_double* %a, align 8                ; <long_double> [#uses=1]
18    %2 = load long_double* %b, align 8                ; <long_double> [#uses=1]
19    %3 = fadd long_double %1, %2                      ; <long_double> [#uses=1]
20    store long_double %3, long_double* %c, align 8
21    %4 = load long_double* %c, align 8                ; <long_double> [#uses=1]
22    %5 = call i32(int) (i8(char)*, ...)* @printf(i8(char)* noalias getelementptr inbounds ([7 x i8(
   char)]* @.str, i32(int) 0, i32(int) 0), long_double %4) nounwind ; <i32(int)> [#uses=0]
23    store i32(int) 0, i32(int)* %0, align 4
24    %6 = load i32(int)* %0, align 4                   ; <i32(int)> [#uses=1]
25    store i32(int) %6, i32(int)* %retval, align 4
26    br label %return
```

Long Double Type

Target dependent Wordcode (x86)

```
 7 define i32() @main() nounwind {
 8 entry:
 9   %retval = alloca i32()                        ; <i32()*> [#uses=2]
10   %c = alloca x86_fp80                          ; <x86_fp80*> [#uses=2]
11   %b = alloca x86_fp80                          ; <x86_fp80*> [#uses=2]
12   %a = alloca x86_fp80                          ; <x86_fp80*> [#uses=2]
13   %0 = alloca i32()                             ; <i32()*> [#uses=2]
14   %"alloca point" = bitcast i32() 0 to i32()    ; <i32()> [#uses=0]
15   store x86_fp80 0xK4000C8F5C28F5C28F800, x86_fp80* %a, align 4
16   store x86_fp80 0xK40008CCCCCCCCCCCD000, x86_fp80* %b, align 4
17   %1 = load x86_fp80* %a, align 4               ; <x86_fp80> [#uses=1]
18   %2 = load x86_fp80* %b, align 4               ; <x86_fp80> [#uses=1]
19   %3 = fadd x86_fp80 %1, %2                     ; <x86_fp80> [#uses=1]
20   store x86_fp80 %3, x86_fp80* %c, align 4
21   %4 = load x86_fp80* %c, align 4               ; <x86_fp80> [#uses=1]
22   %5 = call i32() (i8()*, ...)* @printf(i8()* noalias getelementptr inbounds ([7 x i8()]* @.str,
   i32() 0, i32() 0), x86 fp80 %4) nounwind ; <i32()> [#uses=0]
```

Result

```
jaykang10:~/test$ ./b.org.x86
c=5.340000
jaykang10:~/test$ ./b.mod.opt.x86
c=5.340000
```

Long Double Type

Target dependent Wordcode (ARM)

```
 7 define i32() @main() nounwind {
 8 entry:
 9   %retval = alloca i32()                          ; <i32()*> [#uses=2]
10   %c = alloca double                              ; <double*> [#uses=2]
11   %b = alloca double                              ; <double*> [#uses=2]
12   %a = alloca double                              ; <double*> [#uses=2]
13   %0 = alloca i32()                               ; <i32()*> [#uses=2]
14   %"alloca point" = bitcast i32() 0 to i32()      ; <i32()> [#uses=0]
15   store double 3.140000e+00, double* %a, align 8
16   store double 2.200000e+00, double* %b, align 8
17   %1 = load double* %a, align 8                   ; <double> [#uses=1]
18   %2 = load double* %b, align 8                   ; <double> [#uses=1]
19   %3 = fadd double %1, %2                         ; <double> [#uses=1]
20   store double %3, double* %c, align 8
21   %4 = load double* %c, align 8                   ; <double> [#uses=1]
22   %5 = call i32() (i8()*, ...)* @printf(i8()* noalias getelementptr inbounds ([7 x i8()]* @.str,
     i32() 0, i32() 0), double %4) nounwind ; <i32()> [#uses=0]
```

Result

```
[sbox-FREMANTLE_ARMEL: ~] > ./b.org.arm
c=5.340000
[sbox-FREMANTLE_ARMEL: ~] > ./b.mod.opt.arm
c=5.340000
```

Struct Type with Bitfield

Input

Target Independent Wordcode

Target Dependent Wordcode

ARM

```
struct foo
{
    char a:4;
    long long b:61;
    int c:30;
};
```

⇨

{ i4(char), i61(longlong), i30(int) }

⇨

<{ i8, [7 x i8], i64, i32, [4 x i8]}>

x86

⇨

<{ i8, [3 x i8], i64, i32 }>

```
 1  #include <stdio.h>
 2
 3  struct foo {
 4  char a:4;
 5  long long b:61;
 6  int c:30;
 7  };
 8
 9  struct foo var = {1, 2, 3};
10
11  int main(void) {
12    printf("var.c = %d\n", var.c);
13
14    return 0;
15  }
```

Struct Type with Bitfield

Target independent Wordcode

```
 1  ; ModuleID = 'c.c'
 2  target datalayout = "e-p:32:32:32-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64
    -v64:64:64-v128:128:128-a0:0:64-f192:32:32"
 3  target triple = "ovm-none-linux"
 4
 5  %struct.foo = type { i4(char), i61(longlong), i30(int) }
 6
 7  @var = global %struct.foo { i4(char) 1, i61(longlong) 2, i30(int) 3 } ; <%struct.foo*> [#uses=1]
 8  @.str = private constant [12 x i8(char)] [i8(char) 118, i8(char) 97, i8(char) 114, i8(char) 46, i
    8(char) 99, i8(char) 32, i8(char) 61, i8(char) 32, i8(char) 37, i8(char) 100, i8(char) 10, i8(cha
    r) 0], align 1 ; <[12 x i8(char)]*> [#uses=1]
 9
10  define i32(int) @main() nounwind {
11  entry:
12    %retval = alloca i32(int)                        ; <i32(int)*> [#uses=1]
13    %"alloca point" = bitcast i32() 0 to i32()       ; <i32()> [#uses=0]
14    %0 = ovmresolving load i30(int)* getelementptr inbounds (%struct.foo* @var, i32() 0, i32() 2),
    align 8 ; <i30(int)> [#uses=1]
15    %1 = sext i30(int) %0 to i32(int)                ; <i32(int)> [#uses=1]
16    %2 = call i32(int) (i8(char)*, ...)* @printf(i8(char)* noalias getelementptr inbounds ([12 x i8
    (char)]* @.str, i32(int) 0, i32(int) 0), i32(int) %1) nounwind ; <i32(int)> [#uses=0]
17    br label %return
18
19  return:                                           ; preds = %entry
20    %retval1 = load i32(int)* %retval               ; <i32(int)> [#uses=1]
21    ret i32(int) %retval1
22  }
```

Struct Type with Bitfield

Target dependent Wordcode (x86)

```
3  target triple = "i386-pc-linux-gnu"
4
5  %0 = type { i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i
   8(), i8() }
6  %struct.foo = type <{ i8(), [3 x i8()], i64(), i32() }>
7
8  @var = global %0 { i8() 1, i8() 0, i8() 0, i8() 0, i8() 2, i8() 0, i8() 0, i8() 0, i8() 0, i8() 0
   , i8() 0, i8() 0, i8() 3, i8() 0, i8() 0, i8() 0 } ; <%0*> [#uses=1]
9  @.str = private constant [12 x i8()] c"var.c = %d\0A\00", align 1 ; <[12 x i8()]*> [#uses=1]
10
11 define i32() @main() nounwind {
12 entry:
13   %retval = alloca i32()                          ; <i32()*> [#uses=1]
14   %"alloca point" = bitcast i32() 0 to i32()      ; <i32()> [#uses=0]
15   %0 = load i32()* getelementptr inbounds (%struct.foo* bitcast (%0* @var to %struct.foo*), i32()
   0, i32() 3), align 1 ; <i32()> [#uses=1]
16   %1 = shl i32() %0, 2                            ; <i32()> [#uses=1]
17   %2 = ashr i32() %1, 2                           ; <i32()> [#uses=1]
18   %3 = trunc i32() %2 to i30()                    ; <i30()> [#uses=1]
19   %4 = sext i30() %3 to i32()                     ; <i32()> [#uses=1]
20   %5 = call i32() (i8()*, ...)* @printf(i8()* noalias getelementptr inbounds ([12 x i8()]* @.str,
   i32() 0, i32() 0), i32() %4) nounwind ; <i32()> [#uses=0]
```

Result

```
jaykang10:~/test$ ./c.org.x86
var.c = 3
jaykang10:~/test$ ./c.mod.opt.x86
var.c = 3
```
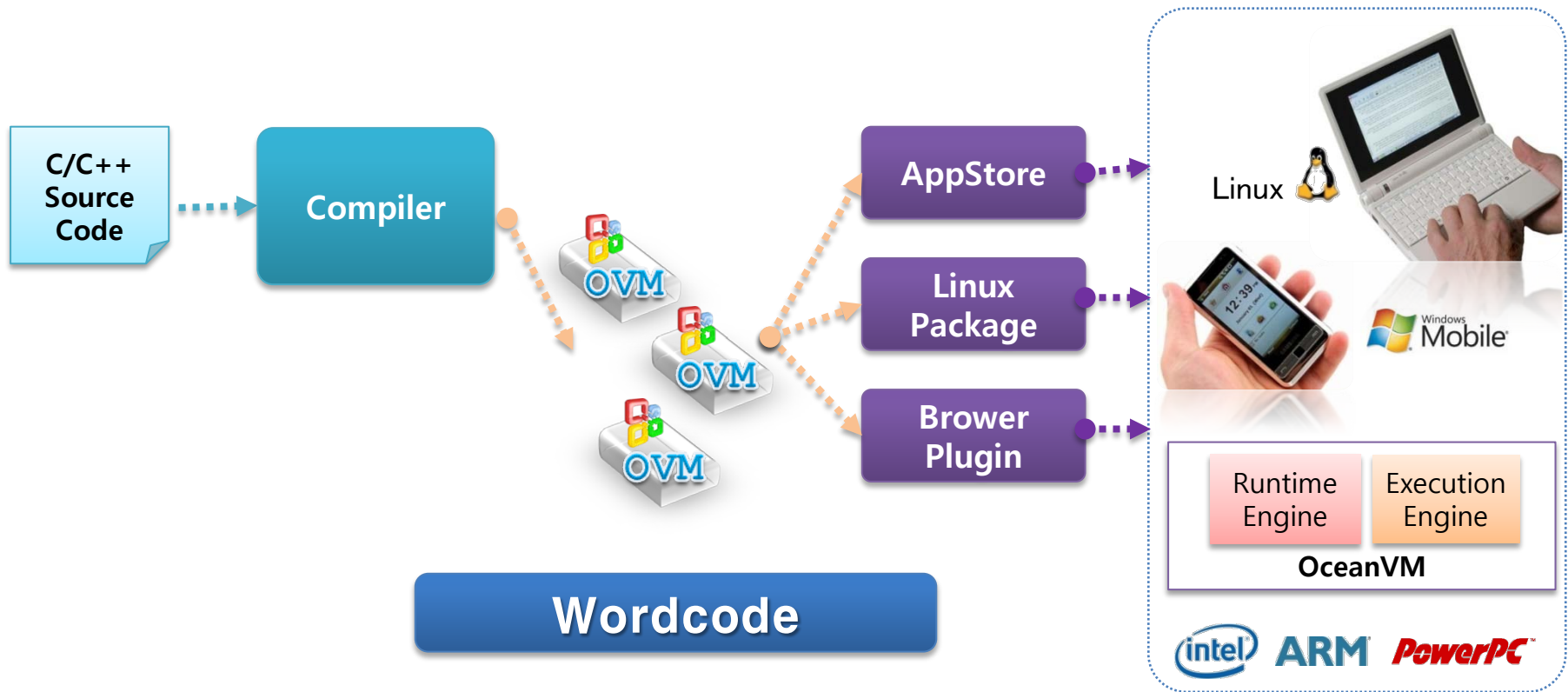
Struct Type with Bitfield

Target dependent Wordcode (ARM)

```
3 target triple = "armv5-none-linux-gnueabi"
4
5 %0 = type { i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i8(), i
   8(), i8(), i8(), i8(), i8(), i8(), [4 x i8()] }
6 %struct.foo = type <{ i8(), [7 x i8()], i64(), i32(), [4 x i8()] }>
7
8 @var = global %0 { i8() 1, i8() 0, i8() 0, i8() 0, i8() 0, i8() 0, i8() 0, i8() 0, i8() 2, i8() 0
   , i8() 0, i8() 0, i8() 0, i8() 0, i8() 0, i8() 0, i8() 3, i8() 0, i8() 0, i8() 0, [4 x i8()] zero
   initializer } ; <%0*> [#uses=1]
9 @.str = private constant [12 x i8()] c"var.c = %d\0A\00", align 1 ; <[12 x i8()]*> [#uses=1]
10
11 define i32() @main() nounwind {
12 entry:
13   %retval = alloca i32()                        ; <i32()*> [#uses=1]
14   %"alloca point" = bitcast i32() 0 to i32()    ; <i32()> [#uses=0]
15   %0 = load i32()* getelementptr inbounds (%struct.foo* bitcast (%0* @var to %struct.foo*), i32()
     0, i32() 3), align 1 ; <i32()> [#uses=1]
16   %1 = shl i32() %0, 2                          ; <i32()> [#uses=1]
17   %2 = ashr i32() %1, 2                         ; <i32()> [#uses=1]
18   %3 = trunc i32() %2 to i30()                  ; <i30()> [#uses=1]
19   %4 = sext i30() %3 to i32()                   ; <i32()> [#uses=1]
20   %5 = call i32() (i8()*, ...)* @printf(i8()* noalias getelementptr inbounds ([12 x i8()]* @.str,
     i32() 0, i32() 0), i32() %4) nounwind ; <i32()> [#uses=0]
```

Result

```
[sbox-FREMANTLE_ARMEL: ~] > ./c.org.arm
var.c = 3
[sbox-FREMANTLE_ARMEL: ~] > ./c.mod.opt.arm
var.c = 3
```

Application

Happy Hacking!