

FileCheck: learning arithmetic

Thomas Preud'homme



GRAPHCORE

Numeric constraints in toolchains

Register constraints

e.g. consecutive 32-bit registers for i64

```
void f(long *ptr) {  
    (...)  
    long res = ptr[0] & ptr[1]  
    (...)  
}
```

and rZ, rX, rY
and r(Z+1), r(X+1), r(Y+1)

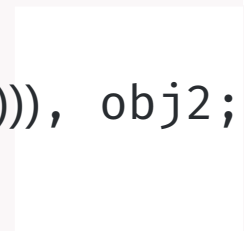
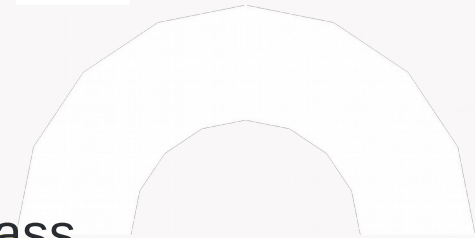
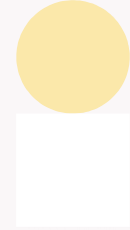
Memory layout

e.g. alignment of fields in struct/class

```
struct foo {  
    int a;  
    long b;  
} obj1, obj2;
```

Addr X : obj1
Addr X+16: obj2

Numeric constraints in toolchains



Register constraints

e.g. consecutive 32-bit registers for i64

```
void f(long a, long b) {  
    return a & b  
}
```

```
CHECK: and    r0, r0, r2  
CHECK: and    r1, r1, r3
```

Memory layout

e.g. alignment of fields in struct/class

```
struct foo {  
    int a;  
    long b;  
} obj1 __attribute__((aligned (256))), obj2;
```

```
CHECK: 0x[[ADDR:[0-9A-F]*]]00: obj1  
CHECK: 0x[[ADDR]]10: obj2
```

Problem: subset of cases tested

FileCheck: introducing arithmetic

Syntax:

```
[[#%fmt,VAR:<relop> expr]]
```

```
[[#%fmt,          VAR+/-num]]
```

```
CHECK: and r[[#X: ]], r[[#Y: ]], r[[#Z: ]]
```

```
CHECK: and r[[#X+1]], r[[#Y+1]], r[[#Z+1]]
```

FileCheck: introducing arithmetic

Syntax:

```
[[#%fmt, VAR:<relop> expr]]
```

```
[[#%fmt, VAR+/-num]]
```

```
CHECK: and r[[#X: ]], r[[#Y: ]], r[[#Z: ]]  
CHECK: and r[[#X+1]], r[[#Y+1]], r[[#Z+1]]
```

```
CHECK: 0x[[#%X, ADDR:      ]] obj1  
CHECK: 0x[[#%X, ADDR + 16 ]] obj2
```

FileCheck: introducing arithmetic

Syntax:

```
[[#%fmt,VAR:  VAR+/-num]]
```

```
CHECK: and r[[#X: ]], r[[#Y: ]], r[[#Z: ]]  
CHECK: and r[[#X+1]], r[[#Y+1]], r[[#Z+1]]
```

```
CHECK: 0x[[#%X,ADDR:      ]] obj1  
CHECK: 0x[[#%X,ADDR + 16  ]] obj2
```

```
return fooba;  
CHECK: file.txt:[[#FOOBA_LINE:@LINE-1]]: unknown variable 'fooba'  
CHECK: file.txt:[[#FOOBA_LINE]]: did you mean 'foobar'
```

FileCheck: introducing arithmetic

Syntax:

```
[[#%fmt, VAR: == expr]]
```

```
Expr operands: + -
```

```
CHECK: and r[[#X: ]], r[[#Y: ]], r[[#Z: ]]  
CHECK: and r[[#X+1]], r[[#Y+1]], r[[#Z+1]]
```

```
CHECK: 0x[[#%X, ADDR:          ]] obj1  
CHECK: 0x[[#%X, ADDR+FOO_SIZE]] obj2
```

```
return fooba;  
CHECK: file.txt: [[#FOOBA_LINE:@LINE-1]]: unknown variable 'fooba'  
CHECK: file.txt: [[#FOOBA_LINE]]: did you mean 'foobar'
```

FileCheck numeric expression: future work

Syntax:

```
[[#%fmt,VAR: == expr]]
```

Expr operands: + -

FileCheck numeric expression: future work

Syntax:

```
[[#%fmt,VAR: == expr]]
```

```
Expr operands: + - * / ( )
```

- Richer expressions

```
CHECK: array size = [[#SIZE: 8*(ELEM_BITSIZE+GAP)]] bytes
```

FileCheck numeric expression: future work

Syntax:

```
[[#%fmt, VAR:<relop> expr]]
```

```
Expr operands: + - * / ( )
```

- Richer expressions

```
CHECK: array size = [[#SIZE: 8*(ELEM_BITSIZE+GAP)]] bytes
```

- Inequalities

```
CHECK: size = [[#SIZE:< 42]] bytes
```

FileCheck numeric expression: future work

Syntax:

```
[[#%fmt, VAR:<relop> expr]]
```

```
Expr operands: + - * / ( )
```

- Richer expressions

```
CHECK: array size = [[#SIZE: 8*(ELEM_BITSIZE+GAP)]] bytes
```

- Inequalities

```
CHECK: size = [[#SIZE:< 42]] bytes
```

- Suggestions? Contribute to llvm-dev ML thread



THANK YOU

Thomas Preud'homme

thomasp@graphcore.ai

