C COMPILER FOR THE MICRO ARCHITECTURE CODE

Shachaf Altman
Gadi Haber
Behnaz Ghouchani

WHAT IS MICRO ARCHITECTURE CODE ?

- Intel CISC CPUs mandate a dual-layer design:
  - External architectural appearance is CISC
  - internal mechanisms employ RISC
- The Internal RISC instructions are called micro-instructions or micro-operations
- Programming of Microcode is done using a low-level microassembly language.

CHALLENGES OF C TO MICROCODE COMPILER

CHALLENGES OF C TO MICROCODE COMPILER

No stack and no static memory

CHALLENGES OF C TO MICROCODE COMPILER

No stack and no static memory

Hybrid calling support : C <-> Microcode

CHALLENGES OF C TO MICROCODE COMPILER

- No stack and no static memory
- Hybrid calling support : C <-> Microcode
- Aggressive optimization of size and performance

CHALLENGES OF C TO MICROCODE COMPILER

- No stack and no static memory
- Hybrid calling support : C <-> Microcode
- Aggressive optimization of size and performance
- Support debugging for aggressively optimized code

CHALLENGES OF C TO MICROCODE COMPILER

- No stack and no static memory
- Hybrid calling support : C <-> Microcode
- Aggressive optimization of size and performance
- Support debugging for aggressively optimized
- Constant changes in micro arch registers should not lead to compiler upgrade

CHALLENGES OF C TO MICROCODE COMPILER

- No stack and no static memory
- Hybrid calling support : C <-> Microcode
- Aggressive optimization of size and performance
- Support debugging for aggressively optimized
- Constant changes in micro arch registers should not lead to compiler upgrade
- Micro arch HW contains large set of restrictions on the usage of registers and instruction groups

BASIC EXAMPLE OF C TO MICROCODE

```
void TEST()
{
 uint32_t x = 1;
 uint64_t y = 0;

  y = x + 6;
  EAX = y;
}
```

BASIC EXAMPLE OF C TO MICROCODE

```
void TEST()
{
 uint32_t x = 1;
 uint64_t y = 0;

  y = x + 6;
  EAX = y;
}
```

```
global TEST():
{
    # uint32_t x = 1;
    R0 := umove32(0x1);

    # uint64_t y = 0;
    R1 := umove64(0x0);

    # y = x + 6;
    R1 := uadd64(R0, 0x6);

    # EAX = y;
    EAX := umove64(R1);
}
```

BASIC EXAMPLE OF C TO MICROCODE

```
void TEST()
{
 uint32_t x = 1;
 uint64_t y = 0;

  y = x + 6;
  EAX = y;
}
```

```
global TEST():
{
    # uint32_t x = 1;
    R0 := umove32(0x1);

    # uint64_t y = 0;
    R1 := umove64(0x0);

    # y = x + 6;
    R1 := uadd64(R0, 0x6);

    # EAX = y;
    EAX := umove64(R1);
}
```

BASIC EXAMPLE OF C TO MICROCODE

```
void TEST()
{
 uint32_t x = 1;
 uint64_t y = 0;

  y = x + 6;
  EAX = y;
}
```

```
global TEST():
{
    # uint32_t x = 1;
    # uint64_t y = 0;
    # y = x + 6;
    # EAX = y;
    EAX := umove64(0x7);
}
```

```
global TEST():
{
    # EAX = y;
    EAX := umove64(R1);
}
```

BASIC EXAMPLE OF C TO MICROCODE

```
void TEST()
{
 uint32_t x = 1;
 uint64_t y = 0;

  y = x + 6;
  EAX = y;
}
```

```
global TEST():
{
    # uint32_t x = 1;
    # uint64_t y = 0;
    # y = x + 6;
    # EAX = y;
    EAX := umove64(0x7);
}
```

```
global TEST():
{
    # EAX = y;
    EAX := umove64(R1);
}
```

SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops

SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops

```
typedef union {
    uint32_t raw32;
    struct {
        uint32_t        x:1;            // reg[0:0]
        uint32_t        field_1:7;      // reg[7:1]
        uint32_t        y:1;            // reg[8:8]
        uint32_t        field_2:22;     // reg[30:9]
        uint32_t        z:1;            // reg[31:31]
    };
} uArchReg_t;

extern uArchReg_t uArchReg;
```

SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops

```
typedef union {
    uint32_t raw32;
    struct {
        uint32_t        x:1;            // reg[0:0]
        uint32_t        field_1:7;      // reg[7:1]
        uint32_t        y:1;            // reg[8:8]
        uint32_t        field_2:22;     // reg[30:9]
        uint32_t        z:1;            // reg[31:31]
    };
} uArchReg_t;

extern uArchReg_t uArchReg;
```

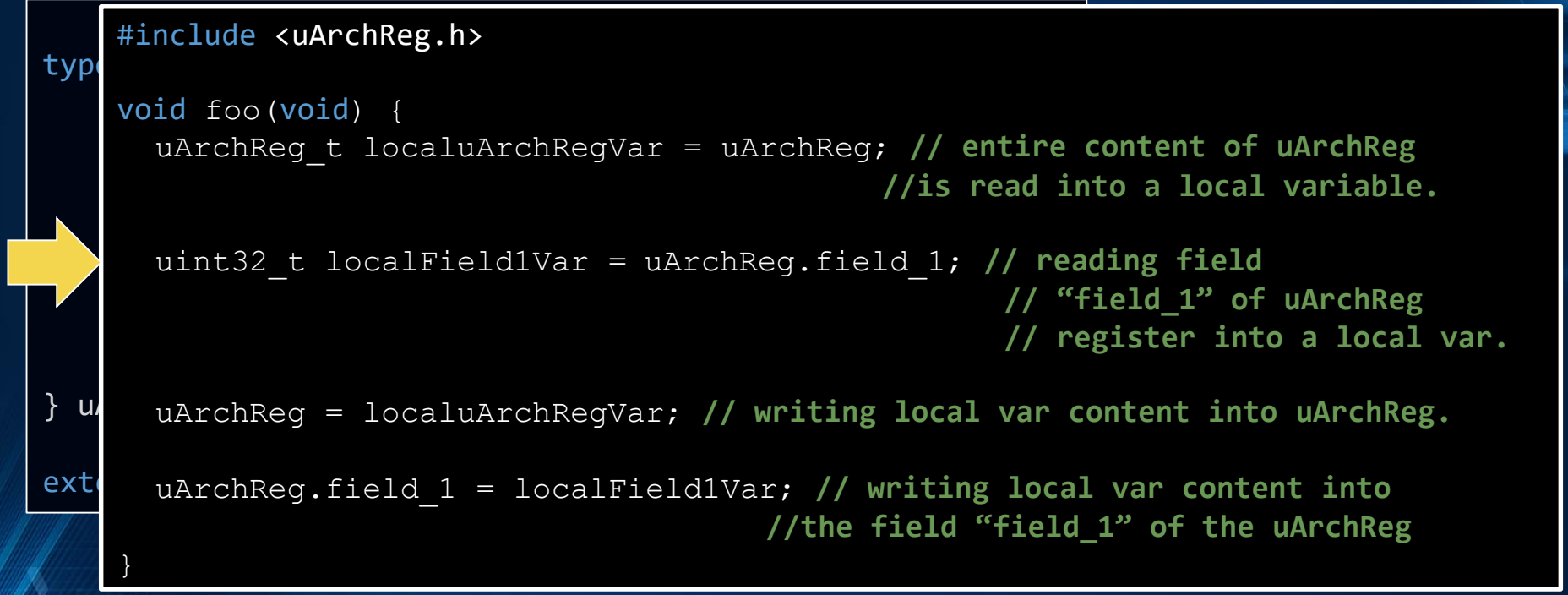
SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops

```
typedef union {
    uint32_t raw32;
    struct {
        uint32_t        x:1;            // reg[0:0]
        uint32_t        field_1:7;      // reg[7:1]
        uint32_t        y:1;            // reg[8:8]
        uint32_t        field_2:22;     // reg[30:9]
        uint32_t        z:1;            // reg[31:31]
    };
} uArchReg_t;

extern uArchReg_t uArchReg;
```

SUPPORTING MICRO-ARCH REGISTERS

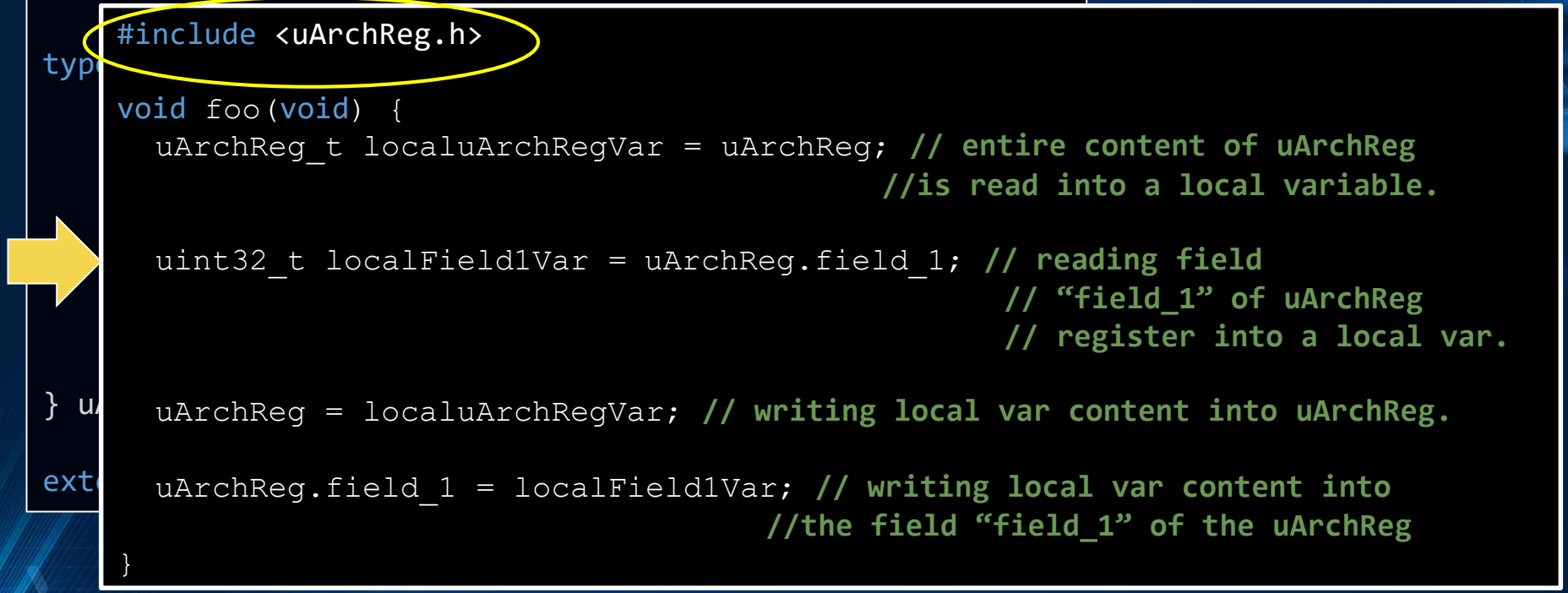
- uArchReg registers are accessed only via special micro-ops



```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops



```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

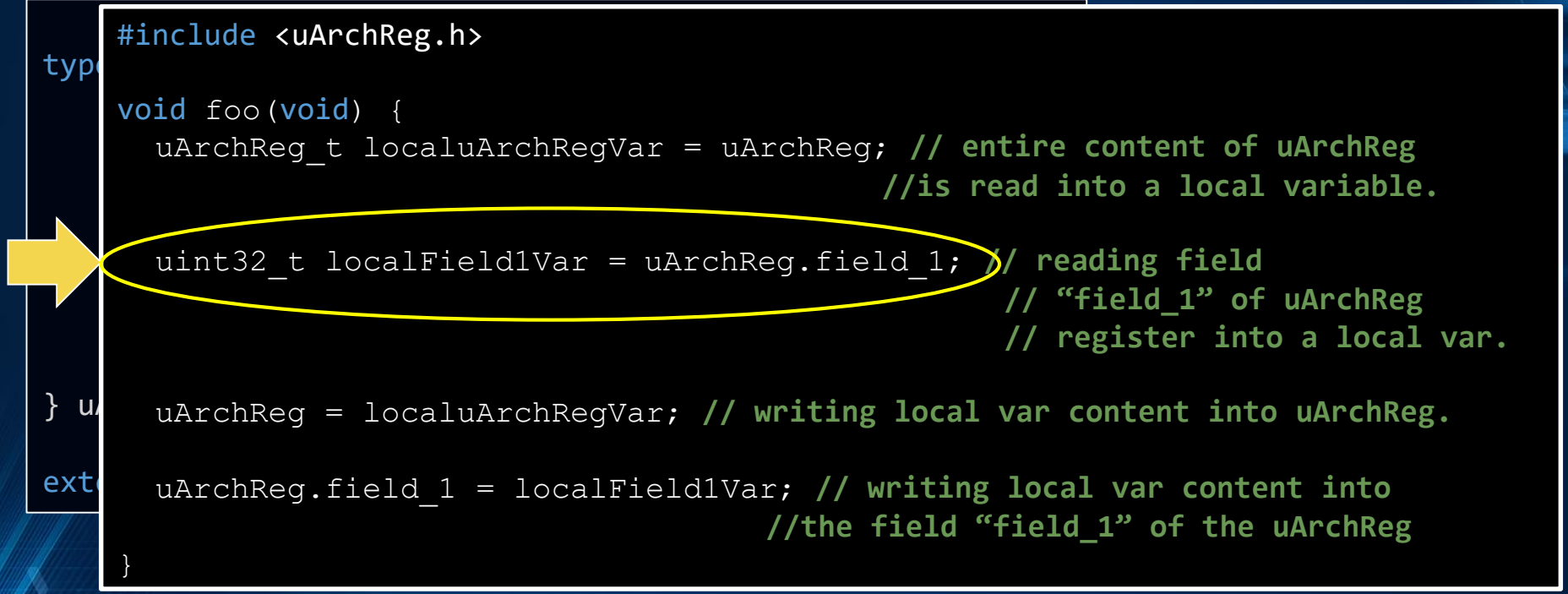
SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops

```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

SUPPORTING MICRO-ARCH REGISTERS

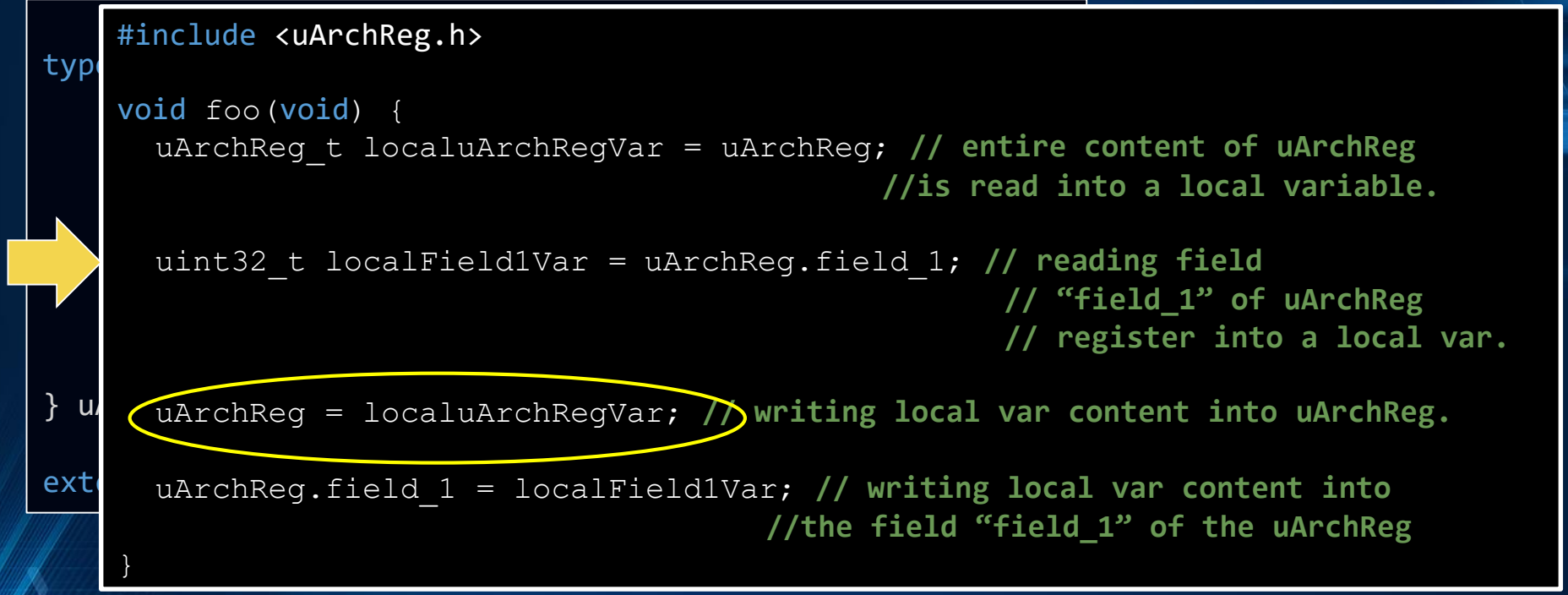
- uArchReg registers are accessed only via special micro-ops



```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

SUPPORTING MICRO-ARCH REGISTERS

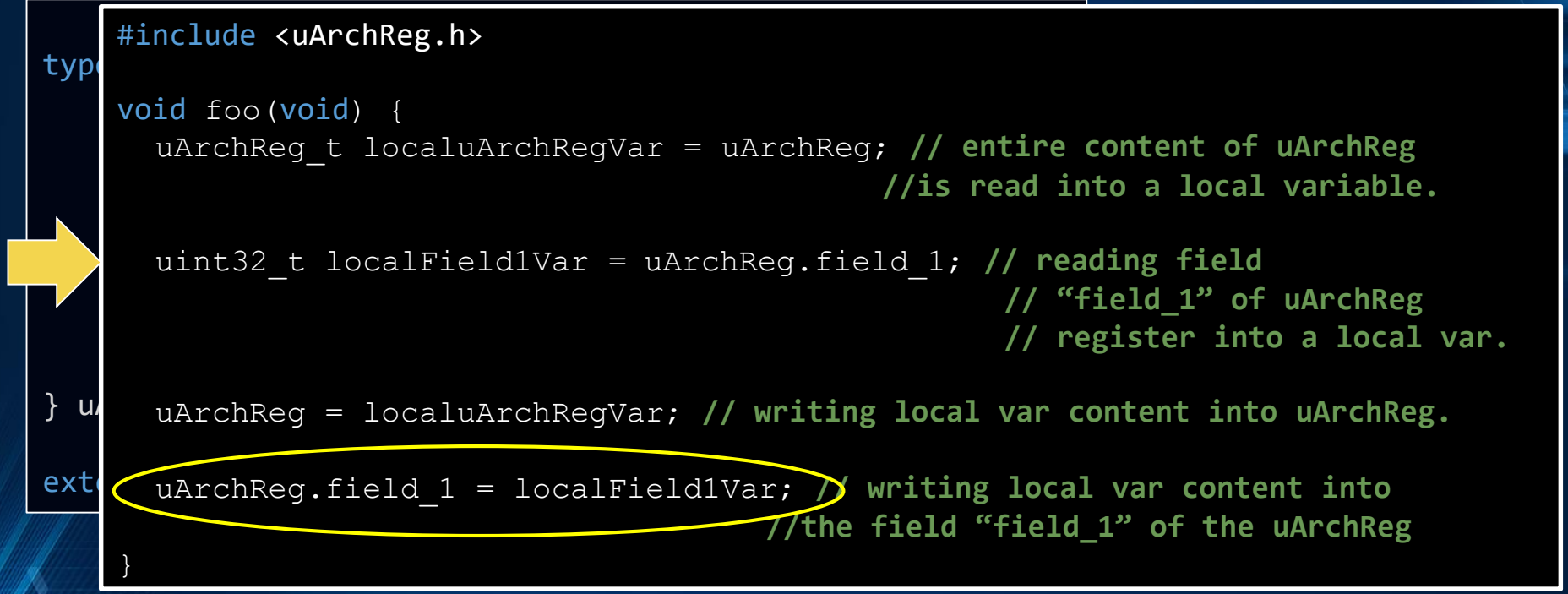
- uArchReg registers are accessed only via special micro-ops



```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

SUPPORTING MICRO-ARCH REGISTERS

- uArchReg registers are accessed only via special micro-ops



```
#include <uArchReg.h>

void foo(void) {
  uArchReg_t localuArchRegVar = uArchReg; // entire content of uArchReg
                                          //is read into a local variable.

  uint32_t localField1Var = uArchReg.field_1; // reading field
                                              // "field_1" of uArchReg
                                              // register into a local var.

  uArchReg = localuArchRegVar; // writing local var content into uArchReg.

  uArchReg.field_1 = localField1Var; // writing local var content into
                                     //the field "field_1" of the uArchReg
}
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if (uArchReg.x & uArchReg.y & uArchReg.z){
      ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if ( uArchReg.x     &     uArchReg.y     &     uArchReg.z){
        ...
```

```
(uArchReg >> x_pose) & 0x1
```

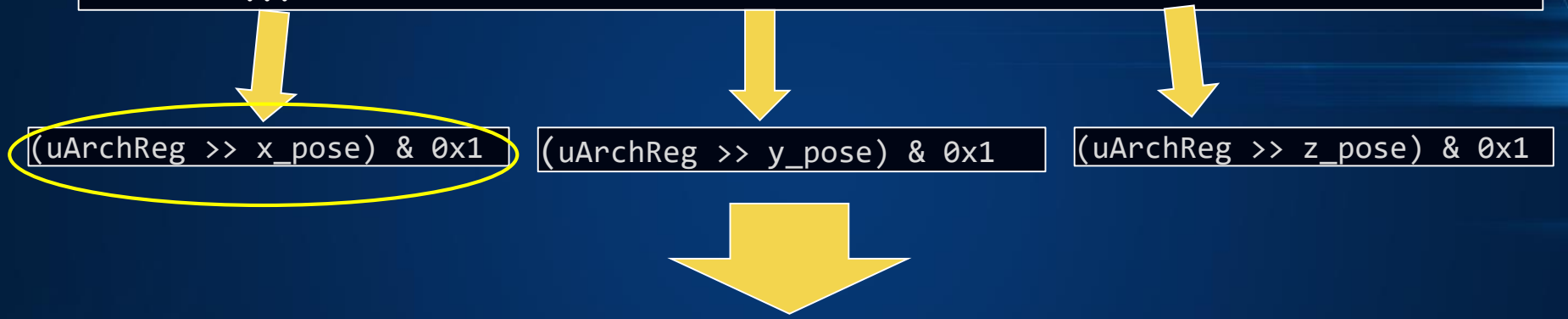
```
(uArchReg >> y_pose) & 0x1
```

```
(uArchReg >> z_pose) & 0x1
```

```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))
        == (1 << x_pose) | 1 << y_pose  | 1 << z_pose))  {
        ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

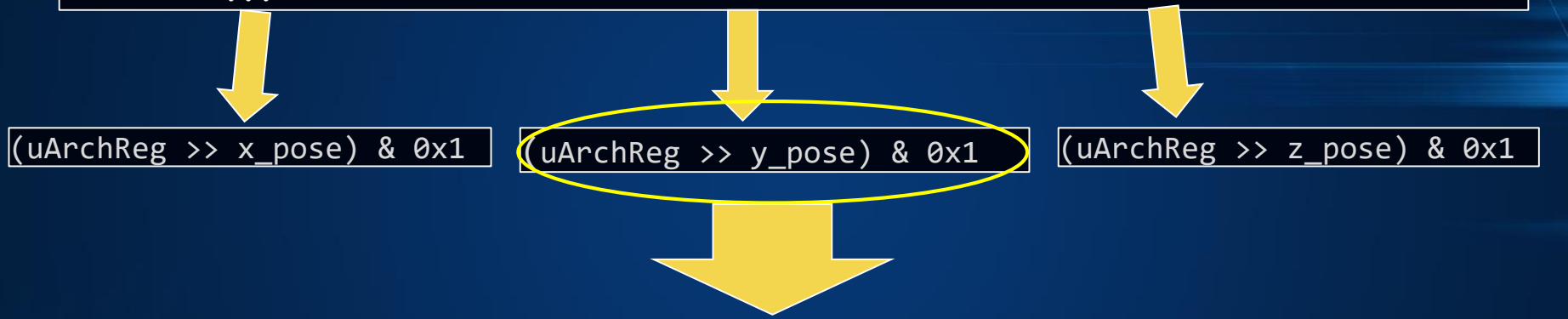
```
if ( uArchReg.x     &     uArchReg.y       &    uArchReg.z){
        ...
```




```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))
        == (1 << x_pose) | 1 << y_pose  | 1 << z_pose))  {
        ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

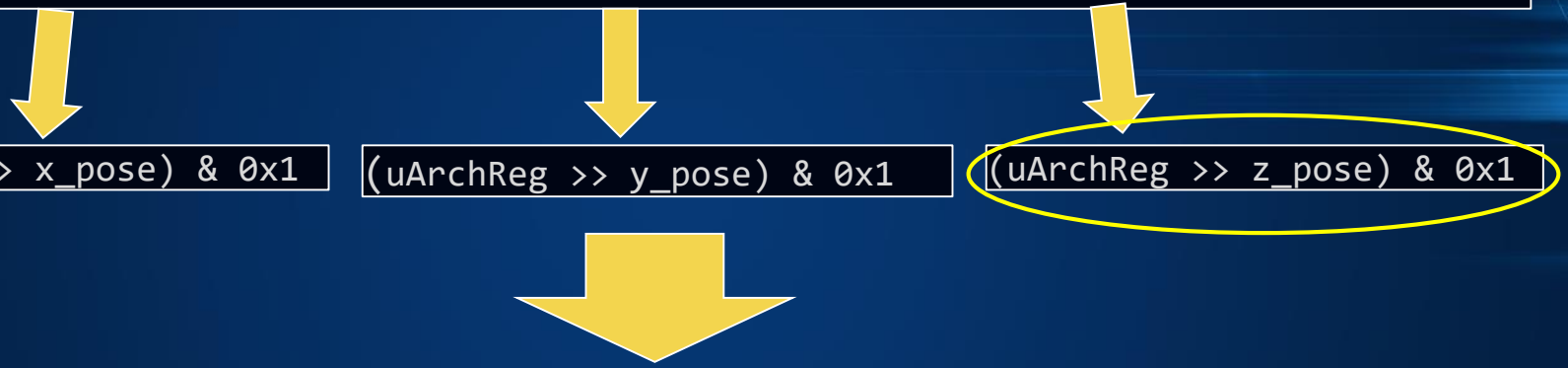
```
if ( uArchReg.x     &     uArchReg.y     &     uArchReg.z){
        ...
```




```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))
        == (1 << x_pose) | 1 << y_pose  | 1 << z_pose))  {
        ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if ( uArchReg.x     &     uArchReg.y     &     uArchReg.z){
        ...
```




```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))
        == (1 << x_pose) | 1 << y_pose  | 1 << z_pose))  {
        ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if ( uArchReg.x     &     uArchReg.y       &    uArchReg.z){
        ...
```

```
(uArchReg >> x_pose) & 0x1
```

```
(uArchReg >> y_pose) & 0x1
```

```
(uArchReg >> z_pose) & 0x1
```

```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))

          == (1 << x_pose) | 1 << y_pose  | 1 << z_pose))      {
       ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if ( uArchReg.x     &     uArchReg.y       &    uArchReg.z){
        ...
```

```
(uArchReg >> x_pose) & 0x1
```

```
(uArchReg >> y_pose) & 0x1
```

```
(uArchReg >> z_pose) & 0x1
```

```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose))

        ==(1 << x_pose) | 1 << y_pose  | 1 << z_pose))  {
        ...
```

TYPICAL OPTIMIZATION ON SINGLE BIT FIELDS

```
if ( uArchReg.x     |     uArchReg.y     |     uArchReg.z){
        ...
```

```
(uArchReg >> x_pose) & 0x1
```

```
(uArchReg >> y_pose) & 0x1
```

```
(uArchReg >> z_pose) & 0x1
```

```
if ((uArchReg & (1 << x_pose | 1 << y_pose | 1 << z_pose)) != 0)    {
                        ...
```

THANK YOU