Compilers Course
Lecture 15: AST to RTL Translation: Data

Array indexing

Given: `ElementType A[N];`
Definition: `&(A[i]) == &A + i * sizeof(ElementType).`

Computing &A

- A is a parameter. Array parameters are passed as pointers, so A should be a pointer value in temp_A: `temp = tempA`
- A is a local variable, in memory at FP+OFFSET_A: `temp = FP+OFFSET_A`
- A is a global variable, in memory at LABEL_A: `temp = LABEL_A`

Computing &(A[i])

```
temp1 = &A (see above)
temp2 = i
temp3 = sizeof(A's element type) // a constant
temp4 = temp2 * temp3 // compute offset to element i
temp5 = temp1 + temp4 // compute actual address
```

Reading the value of A[i]: *(&(A[i]))

```
address = &(A[i]) (see above)
result = load(address)
```

Assigning A[i] = E: *(&(A[i])) = E

```
address = &(A[i]) (see above)
temp = E
store(address, temp)
```

Records: x.field

Definition: `&(x.field) == &x + offsetof(typeof(x), field).`
offsetof(RecordType, Field) is the number of bytes from the start of the *record* to the given *Field.*

```
temp1 = &x
temp2 = temp1 + offset
```

x will generally be a local variable in memory at FP+OFFSET or a global variable in memory at LABEL. Compute *&x* accordingly.

Reading the value of x.field: *(&(x.field))

```
address = &(x.field)
result = load(address)
```

Assigning x.field = E: *(&(x.field)) = E

```
address = &(x.field)
temp = E
store(address, temp)
```

Pointers to record fields: p->field

p->field is defined as (*p).field, so:

```
&(p->field) ==
&((*p).field) ==
&(*p) + offsetof(typeof(*p), field) ==
p + offsetof(typeof(*p), field)
```

so this simply becomes

```
temp1 = p
temp2 = temp1 + offset
```

followed by a load or a store.