Check-in

Review – Function Code Generation

Write X64 code for the following:

```
fn : (int t) int bar {
      t = 7;
}
fn : () int main {
      int a;
      bar(1);
}
```

Announcements & Housekeeping

Administrivia

Review Session

- Monday, Learned 2300 @ 7:00

University of Kansas | Drew Davidson

EECS 665 COMPILER CONSTRUCTION

“Other” Codegen

Last Time

Function Codegen

The Control Flow quads

ifz / goto / nop

Function Parameters and Returns

- System V ABI conventions

<u>You should know</u>

- How to pass primitives arguments
- How to access primitive formals
- How to return primitive values
- How to access return values

Code generation

Stack Alignment

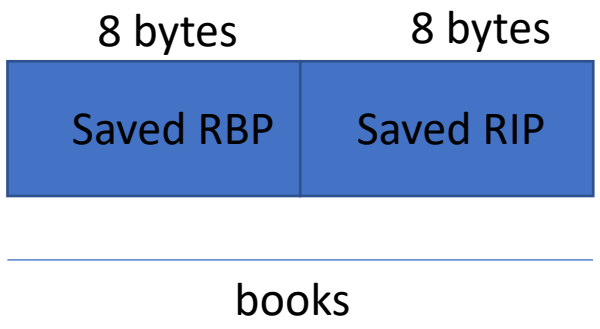
Other Code Generation

System V ABI Assumes %rsp is 16-byte aligned before a call

- Easiest interpretation of this: every AR size should be a multiple of 16

```
void v(){
}
```

```
void a(){
  int v1;
  int v2;
}
```



Stack Alignment

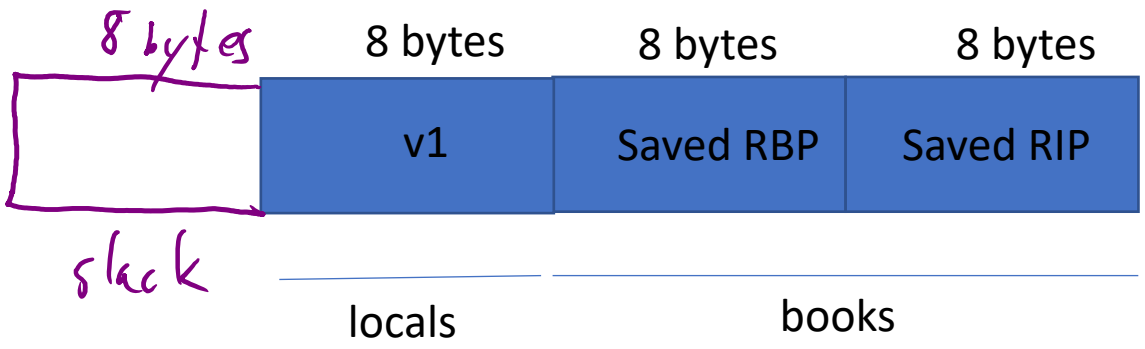
Other Code Generation

System V ABI Assumes %rsp is 16-byte aligned before a call

- Easiest interpretation of this: every AR size should be a multiple of 16

x misaligned!

```
void c(){
  int v1;
}
```



Today's Lecture

Other Code Generation

Other constructs

- Shorter primitive types
- Arrays
- Pointers
- Strings
- Structs

Examples (time permitting)

Machine Codegen

Shorter Primitive Types

Other Code Generation

Recall that the instruction suffix indicates operation size

movq %rax, (gbl_var)

Copy 8 byte from
from all the 8-byte of %rax
to address 8-bytes at address gbl_var

movb %al, (gbl_var)

Copy 1 byte
from lowest byte of A register
to the 1 byte at address gbl_var

movb %rax, (gbl_var)

x nonsense!

Sign Extension

Other Code Generation

```
void foo (){
  short v;
  v = -2S;
}
```

```
[v] := -2
```

```
movw $-2, %ax
movw %ax, -32(%rbp)
movq %rax, -32(%rbp)

movsx %ax, %rbx
movq %rbx, -32(%rbp)
```

Today's Lecture

Other Code Generation

Other constructs

- Shorter primitive types
- Arrays
- Pointers
- Strings
- Structs

Examples (time permitting)

Machine Codegen

Array Codegen

Other Code Generation

Two parts to worry about:

- Data allocation:
  - How will we store an array?
- Code allocation:
  - How are we going to access it?

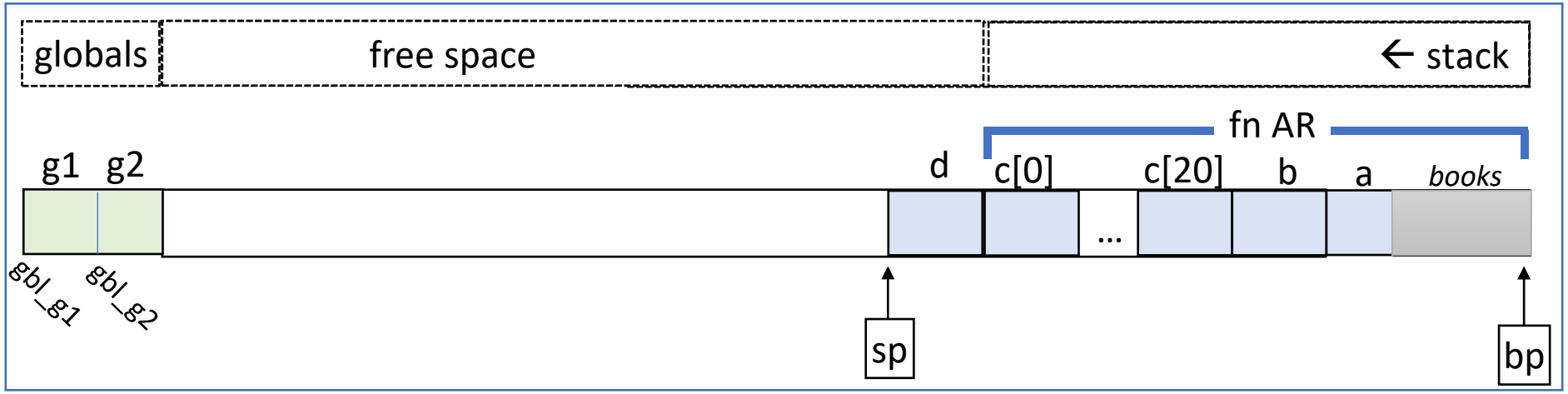
Array Codegen: Data

Other Code Generation

Looks like sequential values in the AR

- Lay out Cell 0 *below* Cell 1
- Access cell i by getting address of cell 0, then adding offset * data type

```
void fn(){
  int a;
  int b;
  int[20] c;
  int d;
}
```



Array Codegen: Code

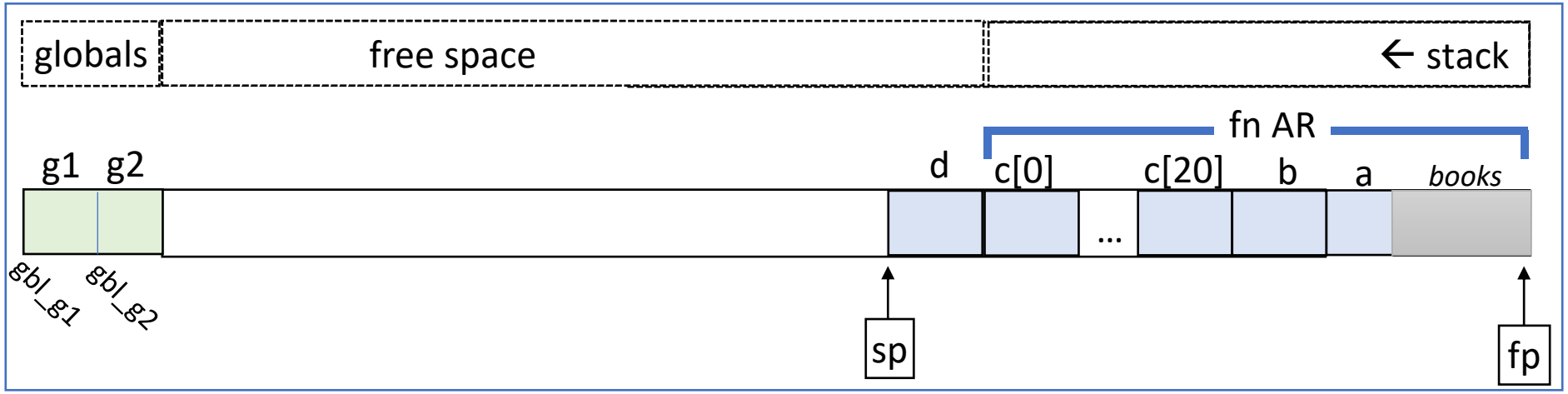
Other Code Generation

We can do math with the array index and value

- Introduce a new Opd type for a memory location

c[3 + a] + b

```
loc1 := c
[tmp1] := 3 + [a]
loc2 := loc1 + [tmp1]
[tmp2] := [loc2]
[tmp3] := [tmp2] + [b]
```



Array Codegen: Code

Other Code Generation

We can do math with the array index and value

- Introduce a new Opd for a memory location
- Need a new set of semantics for operations on a memory location

c[3 + a] + b

```
loc1 := c
[tmp1] := 3 + [a]
loc2 := loc1 + [tmp1]
[tmp2] := [loc2]
[tmp3] := [tmp2] + [b]
```

Array Codegen: Example

Other Code Generation

```
int[3] aG;
fn : () int main {
  int aL[3];
  int idx;
  aG[1] = aL[idx + 1] + 1;
}
```

Today's Lecture

Other Code Generation

Other constructs

- Shorter primitive types
- Arrays
- Pointers
- Strings
- Structs

Examples (time permitting)

Machine Codegen

Pointer Codegen

Other Code Generation

Once you've got arrays, you've got pointers

- Operations on a pointer are just like operations on an array index
  - Need to account for the data type
  - May add bounds checking depending on the language

Dereference local int64_t a	[tmp1] = [[a]]	movq -24(%rbp), %rax
		movq (%rax), %rbx
		movq %rbx, -32(%rbp)

Dereference global int8_t b	[tmp2] = [[b]]	movq (gbl_b), %rax
		movb (%rax), %bl
		movb %bl, -32(%rbp)

Today's Lecture

Other Code Generation

Other constructs

- Shorter primitive types
- Arrays
- Pointers
- Strings
- Structs

Examples (time permitting)

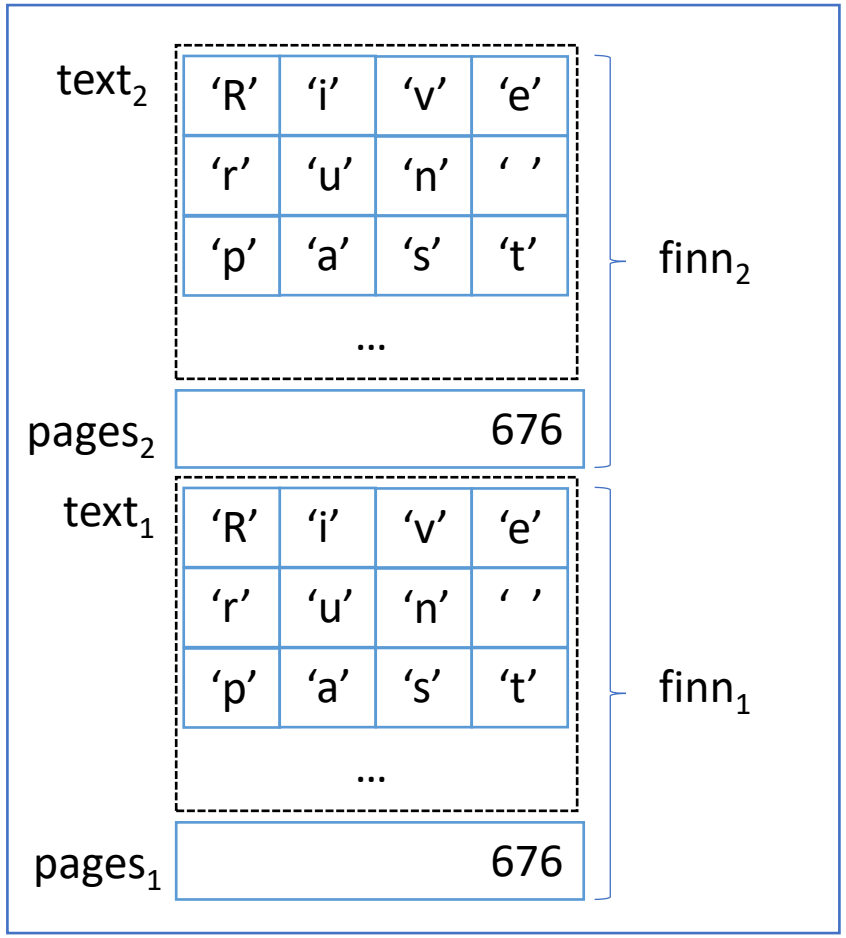
Machine Codegen

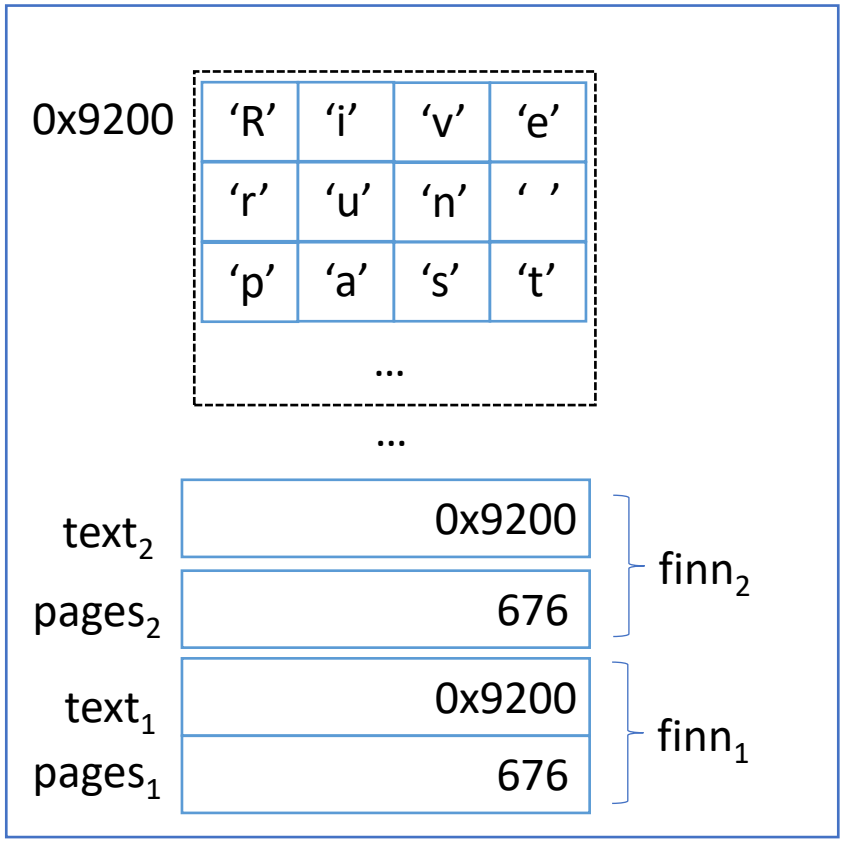
String Codegen

Other Code Generation

Put static data in global memory (e.g. .asciz)

```
finn(){
  char * text = "Riverrun past[…]";
  int pages = 676;
  finn();
}
```





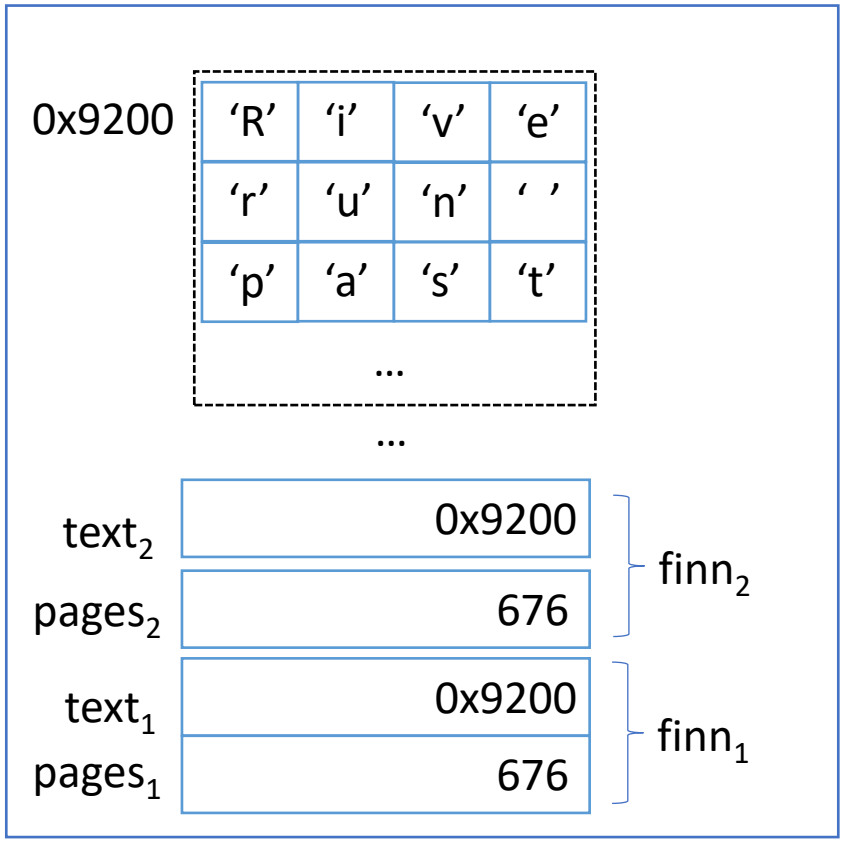
String Codegen

Other Code Generation

Put static data in global memory (e.g. .asciz)

```
.data
str_text:
    .asciz "Riverrun past[…]";
len_text:
    .quad 700000
…
.text
finn:
    …
    movq $str_text, %rdi
    movq (len_text), %rsi
    movq $0, $rax
    syscall
    …
```

```
finn(){
  char * text = "Riverrun past[…]";
  int pages = 676;
  finn();
}
```



Today's Lecture

Other Code Generation

Other constructs

- Shorter primitive types
- Arrays
- Pointers
- Strings
- Structs

Examples (time permitting)

Machine Codegen

Struct/Class Handling

Code Generation III (Other Codegen)

Basic idea:

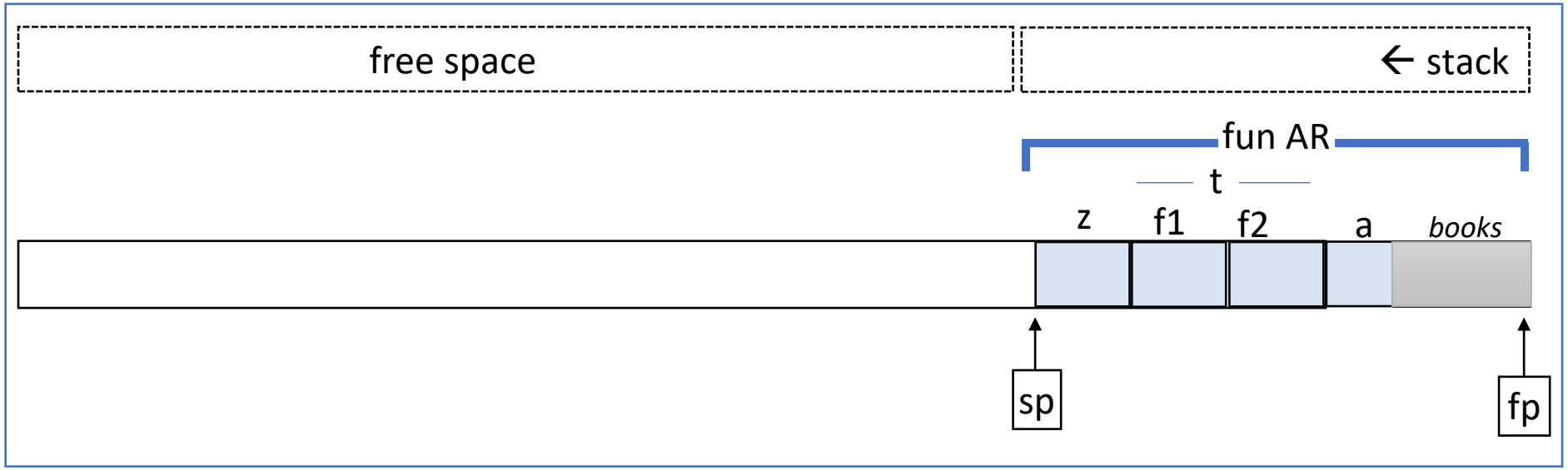
- Flatten all fields, lay out sequentially in memory

```
struct TwoInts {
    int f1;
    int f2;
}
void fun(){
    int a;
    struct TwoInts t;
    int z;
}
```

Struct/Class Handling

Code Generation III (Other Codegen)

```
struct TwoInts {
    int f1;
    int f2;
}
void fun(){
    int a;
    struct TwoInts t;
    int z;
}
```



Struct/Class Dot Access

Code Generation III (Other Codegen)

- Fortunately, we know the offset from the base of a struct to a certain field statically
  - The compiler can do the math for the slot address

  (In contrast with pointers)

```
struct Demo inst;
struct Demo inst2;
inst.b.c = inst2.b.c + 1;
```

load this address load this value

```
struct Inner{
  bool hi;
  int there;
  int c;
};
struct Demo{
  struct Inner b;
  int val;
};
```

What About Our Compiler?

Other Code Generation: Addendum

- Much of the preceding discussion won't be necessary for the projects:
  - We **do** have strings and classes
  - Our 3AC already expects that strings will be global

Lecture Outline

Other Code Generation

Other constructs

- Scopes
- Arrays
- Pointers
- Strings
- Structs

Not an exhaustive list!

Examples

Machine Codegen

Summary

Other Code Generation

Best practices for the language depend on the constructs

- Helpful to have a notion of memory address
- Immutable strings means they can be global

Next Time

Other Code Generation

- We'll look at optimizing the machine code

(BONUS MATERIAL)

Anything after this slide is bonus material and will not be on exams / projects / written work

Accessing Outer Scopes

Other Code Generation

- Static scope
  - Variable declared in one procedure and accessed in a nested one
- Dynamic scope
  - Any variable use not locally declared

Nested Functions

Code Generation III (Other Codegen)

- Each function has it's own AR
  - Inner function accesses the outer AR

```
function main(){
    int a = 0;
    function subprog(){
        a = a + 1;
    }
}
```

Static Scope, Non-Local Access

Code Generation III (Other Codegen)

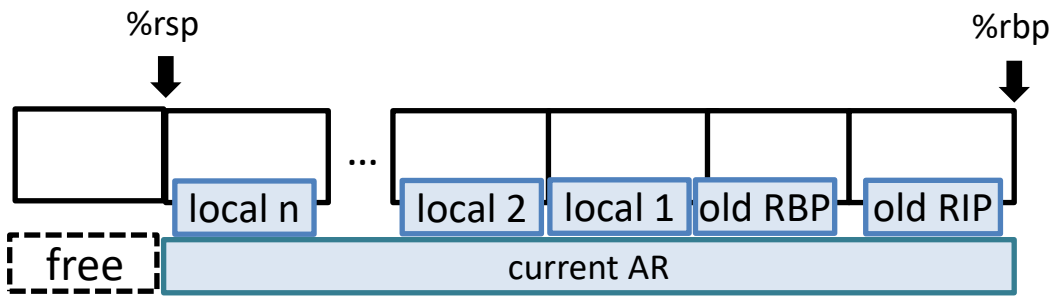
```
void procA(){  // level 1
  int x, y;
  void procB(){ // level 2

    void procC(){ //level 3
      int z;
      void procD(){
        int x;
        x = z + y;
        procB();
      }
      x = 4;
      z = 2;
      procB();
      procD();
    }
    x = 3;
    y = 5;
}
```

Access Links

Code Generation III (Other Codegen)

Add an additional field to the AR

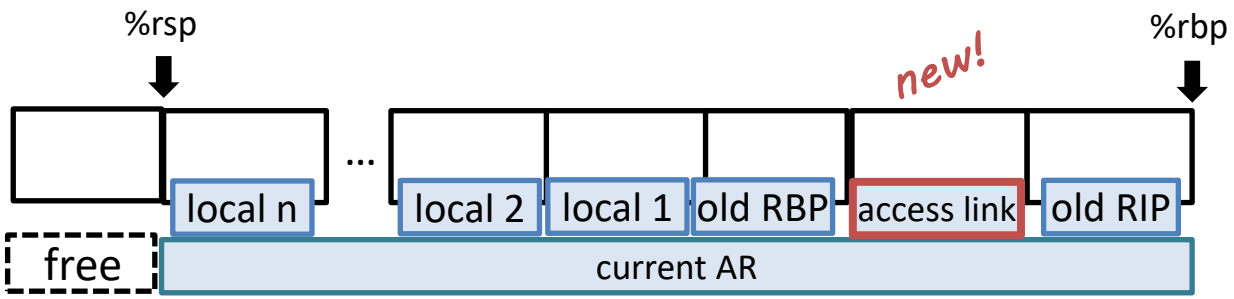


Access Links

Code Generation III (Other Codegen)

Add an additional field to the AR

- Points to the locals area of the outer function
- Sometimes called the static link (since it refers to the static nesting)



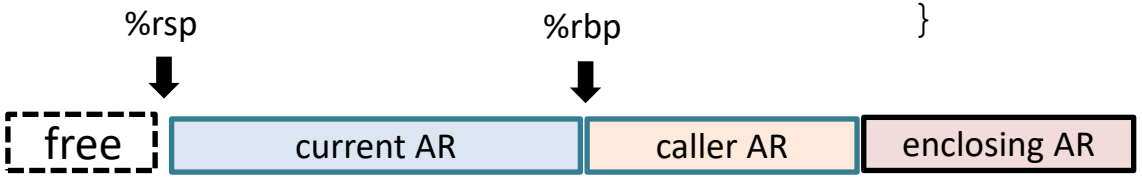
Access Links

Code Generation III (Other Codegen)

Add an additional field to the AR

- Points to the locals area of the outer function
- Sometimes called the static link (since it refers to the static nesting)
- NOT NECESSARILY the caller AR

```
void procA(){
  int a1;
  void procB(){
  }
  void procC(){
    int c1;
    procB();
  }
}
```



How Access Links Work

Code Generation III (Other Codegen)

- We know how many *levels* to traverse statically
  - Example: In nesting level 3 and the variable is in nesting level 1: go back access links

    (3 – 1) 2 levels

Traversing Access Links

Code Generation III (Other Codegen)

Using 1 access link

```
movq -16(%rbp), %rax
movq -32(%rax), %rax
```

Using 4 access links

```
movq -16(%rbp), %rax
movq -16(%rax) %rax
movq -16(%rax) %rax
movq -32(%rax), %rax
```

Thinking About Access Links

Code Generation III (Other Codegen)

- We know the variable we want to access statically
  - Why don't we just index into the parent's AR using a large positive offset from $fp?

```
movq 38(%rbp) %rax
```