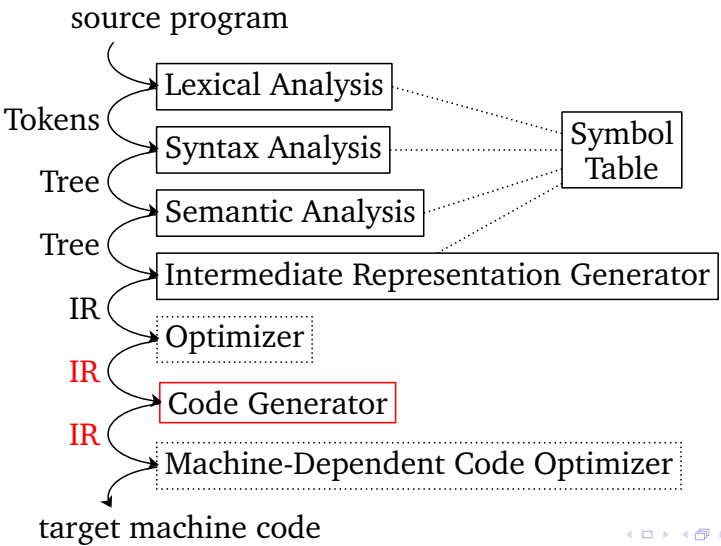
# Runtime Environments

Eva Rose Kristoffer Rose

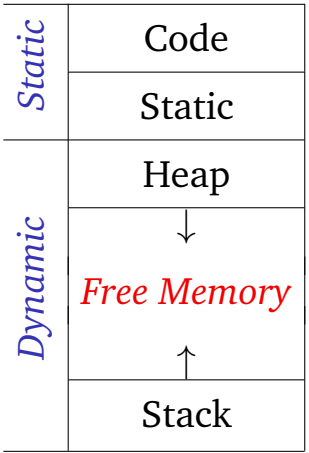
NYU Courant Institute
Compiler Construction (CSCI-GA.2130-001)
http://cs.nyu.edu/courses/fall14/CSCI-GA.2130-001/lecture-8.pdf

October 30, 2014

# Sixth compilation phase



# Storage Organization



# Storage Organization

| | |
|---|---|
| Code area | instructions, address ptrs. |
| Static area | global constants, internalized strings. |
| Heap | objects, records, arrays, variable strings. |
| Runtime stack | activation records/stack frames. |

# Activation trees

```
...
void quicksort(int m, int n){
    int i;
    if (n > m){
        i = partition(m,n); /* picks "quicksort separator value" */
        quicksort(m, i-1);
        quicksort(i+1, n);
    }

main() {
    readArray();  /* reads 9 integers into a[1],..,a[9] */
    ...           /* sets sentenials a[0], a[10] */
    quicksort(1,9);
}
```

# Activation trees

*Figure 7.4 (p.433):*

*Show activation tree, and*
*show live activations when control is with quicksort(2,3)*

# The Stack

Consider the (pseudo) program:

```
main() { f(); g(); }

f() { if ... then f(); else g(); }

g() {...}
```

*Show all states of runtime stack activations during execution.*
*Compare with live activations in associated activation tree.*

# The Stack

Consider the (pseudo) program:

```
main() { f(); g(); }

f() { if ... then f(); else g(); }

g() {...}
```

*Show all states of runtime stack activations during execution.*

*Compare with live activations in associated activation tree.*

# The Stack

Consider the (pseudo) program:

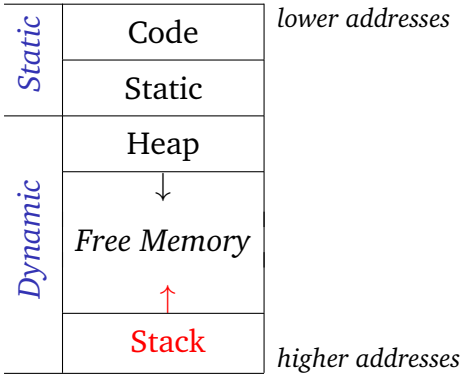
```
main() { f(); g(); }

f() { if ... then f(); else g(); }

g() {...}
```

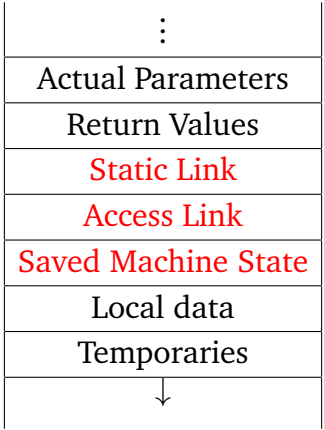
*Show all states of runtime stack activations during execution.*
*Compare with live activations in associated activation tree.*

# The Stack


Static
Code
Static
Dynamic
Heap
Free Memory
Stack
lower addresses
higher addresses

# The Stack

Activation record or “frame”:



# The Stack

Typical “frame core” scenario:

Static link (saved frame pointer, or control link)
points to saved “frame core” (of caller).

Access link for nested procedures, typically absent.

Saved machine status saved PC and registers (of caller).

# Target Arhitectures

High-end (expensive) architectures:

- ARM: Tablets, mobile phones, ultrabooks, Raspberry Pi.
- amd64: Intel (and AMD) PCs and servers.
- SPARC: Oracle servers.
- Power PC: IBM servers.

Low-end (cheap) architectures:

- 6505: 8-bit architecture...

# The Stack

Calling Conventions generally speaking:

*A "static storage allocation contract" between caller and callee, which is guaranteed to hold at runtime.*

- size of allocated entities (actual parameters and return values, e.g. based on static type info),
- where address information is kept (registers or stack).

# ARM Calling Conventions

| *Register* | *On Entry* | *On Return* |
|---|---|---|
| r0–3 | parameter or unused | return value or unused |
| r4–11 | preserved | same as on entry |
| r12 | undefined | undefined |
| r13 ‘sp’ | stack pointer | same as on entry |
| r14 ‘lr’ | return address | (unconstrained) |
| r15 ‘pc’ | program counter | return address |

*For full details see*
*http:*
*//infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf*

# ARM Calling Conventions

| *Register* | *On Entry* | *On Return* |
|---|---|---|
| r0–3 | parameter or unused | return value or unused |
| r4–11 | preserved | same as on entry |
| r12 | undefined | undefined |
| r13 ‘sp’ | stack pointer | same as on entry |
| r14 ‘lr’ | return address | (unconstrained) |
| r15 ‘pc’ | program counter | return address |

*For full details see*
`http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf`

# ARM Instructions

- `MOV` $B$`,` $A$ – "*Move*"
  copy what is in $A$ to $B$.
- `BL` *label* – "*Branch with Link*"
  set lr to next instruction address then jump to *label*.
- `STMFD sp!,{`*regs*`}` – "*Store Multiple/Fully Descending*"
  push all the listed *regs* on stack (that grows downwards).
- `LDMFD sp!,{`*regs*`}` – "*Load Multiple/Fully Descending*"
  pop all the listed *regs* from stack (downwards).

# ARM Call and Return Code

```
Caller:
  MOV r0, <param1>
  MOV r1, <param2>
  MOV r2, <param3>
  MOV r3, <param4>
  BL Callee
```

```
Callee:
  STMFD sp!,{r4-r11,lr}

  Body with parameters in r0-r3...

  MOV r0, <return value>
  LDMFD sp!,{r4-r11,pc}
```

```
  Use return value in r0
```

# AMD64 (Intel) Call and Return Code

```
Caller:
  put actuals on stack/registers
  call  Callee
                          Callee:
                            push %rbp
                            move %rbp, %rsp
                            sub %rsp, <frame size>

                            Body with parameters in [%rbp-...]...

                            move %rax, <return value>
                            move %rsp, %rbp
                            pop %rbp
                            ret
  remove actuals from stack
  Use return value in %rax
```

# Other stack topics

- saving registers across calls,
- variable-sized data on stack,
- accessing non-local stack variables.

# ARM Call and Return Code Example

```
Caller:
  MOV r0, #1
  MOV r1, #1
  BL Callee

  Continue...
```

```
Callee:
  STMFD sp!,{r4-r11,lr}
  ADD r0, r0, r1
  LDMFD sp!,{r4-r11,pc}
```

# ARM Call and Return Code Example

```
Caller:
  MOV r0, #1
  MOV r1, #1
  BL Callee

  Continue...
```

```
Callee:
  STMFD sp!,{r4-r11,lr}
  ADD r0, r0, r1
  LDMFD sp!,{r4-r11,pc}
```

# ARM Call and Return Code Example

```
Caller:
  MOV r0, #1
  MOV r1, #1
  BL Callee

  Continue...
```

```
Callee:
  STMFD sp!,{lr}
  ADD r0, r0, r1
  LDMFD sp!,{pc}
```

# ARM Call and Return Code Example

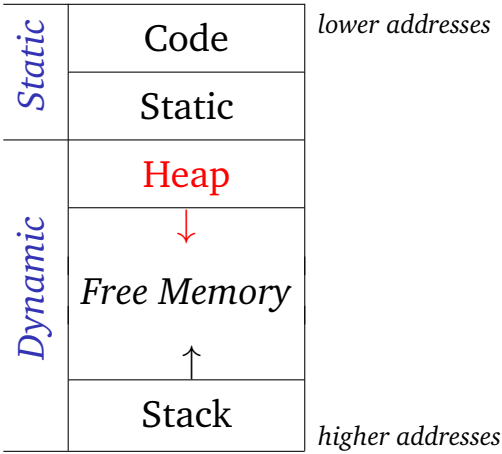
```
Caller:
  MOV r0, #1
  MOV r1, #1
  BL Callee

  Continue...
```

```
Callee:

  ADD r0, r0, r1
  MOV pc, lr
```

# The heap


Static
Code
Static
Dynamic
Heap
Free Memory
Stack
lower addresses
higher addresses

# The heap

*Heap "objects" may live indefinitely.*
*“Life span” is managed automatically or from program (memory manager).*

| | **Java** | **C** | **C++** |
|---|---|---|---|
| **Allocation** | `new` | `malloc` | `new` |
| **Deallocation** | *garbage c.* | `free` | `delete` |

*Questions?*