Introduction to Computer Engineering

CS/ECE 252, Spring 2017
Rahul Nayar
Computer Sciences Department
University of Wisconsin – Madison
Stack and Procedures
Why Stack?

- Running nested procedure calls in assembly language
  - for each procedure calls return address stored in R7

- How do we keep track of the program execution in nested procedure calls?
Stack:

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages.
- It follows LIFO principle to access the data values
- **LIFO**: Last in first out
- Two basic operations:
  - PUSH
  - POP
Stack:

Push

Last In - First Out

Pop

Stack

Stack
Memory layout

$2^N - 1$

- Stack: local variables and procedure context
- Dynamic Data (Heap): new allocated variables
- Static Data: static variables
- Literals
- Instructions
Memory layout

- Stack: Managed “automatically” (by compiler)
- Dynamic Data (Heap): Managed by programmer
- Static Data: Initialized when process starts
- Literals: Initialized when process starts
- Instructions: Initialized when process starts
IA32 Class Stack: Push

- push SRC
IA32 Class Stack: Push

- push SRC
  - Fetch value from SRC
  - Decrement %esp by 4 (why?)
  - store value at address given by
    ➢ %esp
IA32 Class Stack: Pop

- pop SRC
  - Load value from address %esp
  - write value to Dest
  - Increment %esp by 4
Procedure calls

- Callee must know where to find the args
- Callee must know where to find the “return address”
- Caller must know where to find the return value
- Caller and Callee run on the same CPU. use the same registers
  - Caller might need to save the register that the Callee might use
  - Callee might need to save the registers that the Caller had used
Procedure calls

- The convention of where to leave/find things is called the procedure call linkage
  - Details vary between different systems
  - What could happen if our program did not follow these conventions?

![Diagram showing caller and callee process flow](image)
Procedure calls: Control flow

- Use stack to support procedure call and return
- Procedure call: call label
  - Push return address on stack
  - Jump to label
- call label PC=x4395

```
  esp  x123
  x3213
  x4395
```
Procedure calls: Control flow

- Use stack to support procedure call and return
- Procedure return: ret %esp
  - Pop return address from stack
  - Jump to address set from the pop instruction
- set PC=x4395
Return values convention

• This is a convention to store return values
  • this is not necessary
  • Done to make the writing programs easier

• IA32 return values are stored in %eax register
  • Choice is arbitrary and could have easily been a different register

• Caller must make sure to save this register value before calling a callee that returns a value

• Callee placed return value in %eax register

• Upon return, caller finds the return value from the %eax register
Stack Frames

- Contents of a stack frame
  - Local variables
  - Function arguments
  - Return Information
  - Temporary Space
Stack Frames

- Management
  - Space allocated when procedure is entered
    - Set up code
  - Space deallocated upon return
    - Finish code
Stack frame structure
Call Chain Example

```
yoo(...) {
  ...
  who();
  ...
}
```

```
who(...) {
  ...
  amI();
  ...
  amI();
  ...
}
```

```
amI(...) {
  ...
  ...
  amI();
  ...
}
```

Procedure amI is recursive (calls itself)
Call Chain Example

```c
yoo(...) {
    ...
    who();
    ...
}
```
Call Chain Example

```c
who(...) {
    * * *
    amI();
    * * *
}
```

```c
yoo

who

%ebp

%esp

amI

amI

amI

amI

amI

yoo

who
```
Call Chain Example

```c
amI(...) {
    ...
    amI();
    ...
}
```
Call Chain Example
Call Chain Example

```c
amI(...) {
    ...
    amI();
    ...
}
```
Call Chain Example

```c
amI(...) {
    ...
    amI();
    ...
}
```

Diagram of call chain with function calls and stack frames.
Call Chain Example

```
amI(...) {
    ...
    amI();
}
```
Call Chain Example
Call Chain Example
Call Chain Example
Call Chain Example
Stack frame structure