Machine-Level Programming III:

Procedures.

(Application Binary Interfaces)

ABI

Mechanisms in procedures

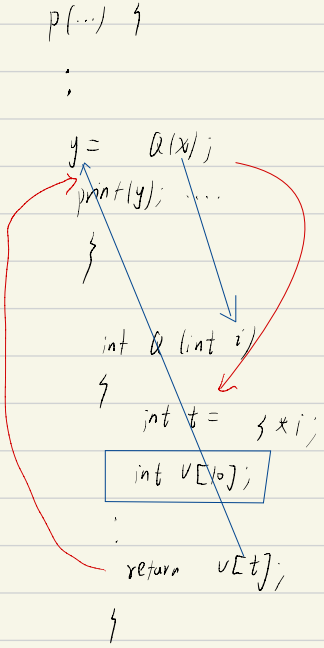
(I) passing control
- (i) To beginning of procedure code
- (ii) Back to return point.

(II) Passing data
- (i) Procedure arguments
- (ii) return values

(III) Memory management.
- (i) Allocate during procedure execution
- (ii) Deallocate upon return.
  - just move the stack pointer

(IV) Mechanisms all implemented with machine instructions

(V) X86-IA implementation of a procedure uses only those mechanisms required.



programming level: ⟵ memory is just a great array of bytes.

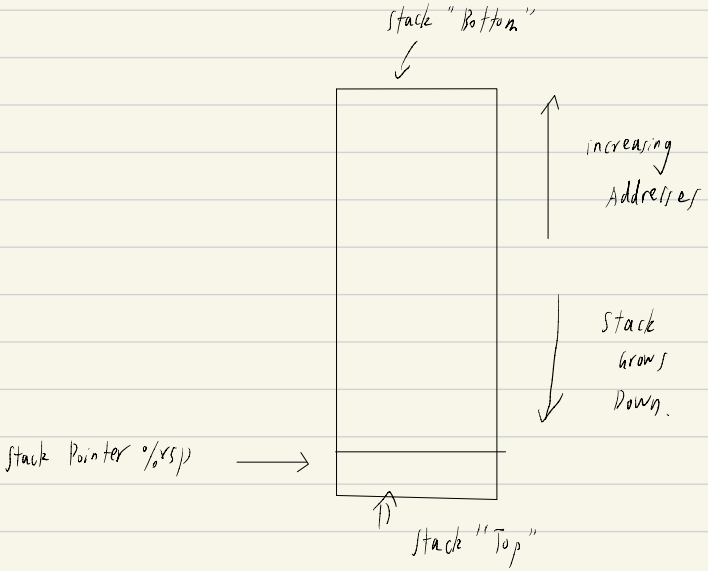
x86-64 stack

(I) Region of memory managed with stack discipline

(II) Grows toward lower addresses

(III) Register %rsp contains lowest stack address

(i) address of "top" element.



(IV) pushq Src.

(i) Fetch operand at Src.

(ii) Decrement %rsp by 8

(iii) write operand at address given by %rsp.

(V) popq Dest:

(i) Read value at address given by %rsp.

(ii) increment %rsp by 8

(iii) Store value at Dest (must be register)

see (multstore_mult2.txt
multistore_mult2.c)

(I) Use stack to support procedure call and return.

(II) Procedure call: call label:

(i) Push return address on stack

(ii) Jump to label

(III) Return address:

(i) Address of the next instruction right after call

(ii) Example from disassembly

(IV) procedure return: ret

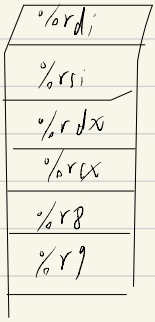
(i) Pop address from stack

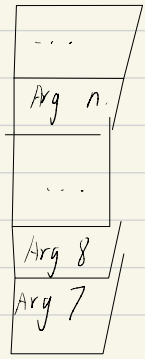
(ii) Jump to address

Procedure Data Flow

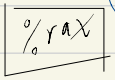
Registers (Assume only pointer and int data)

First 6 arguments





Return value (stored at)



- Only allocate stack space when needed.

Register faster >> access to the memory

all addresses are 8-bytes unit.

Stack-Based Languages

(i) Languages that support recursion.

(ii) C, Pascal, Java

(iii) Code must be "Reentrant"
multiple simultaneous instantiations of single procedure.

(iv) Need some place to store state of each instantiation.
- Arguments
- Local variables
- Return pointer.

(v) Stack discipline

(i) State for given procedure needed for limited time

(ii) From when called to when return

(iii) Callee returns before caller does

(iv) Stack allocated in Frames

(i) state for single procedure instantiation.

Stack frame: each block we use for a particular call.

you
who
am I
am I
am I
Stack
call
return
you
who
am I
am I
%rbp
am I
%rsp

x86-64 / Linux Stack Frame

(I) Current Stack Frame ("Top" to Bottom)

(i) "Argument build":
Parameters for function about to call

(ii) Local variables:
If can't keep in registers

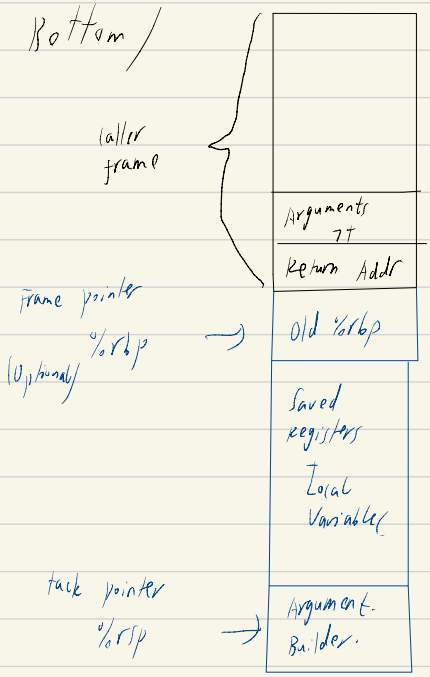
(iii) Saved register context

(iv) Old frame pointer (optional)

(II) Caller Stack Frame.

(i) Return address
- Pushed by call instruction.

(ii) Arguments for this call



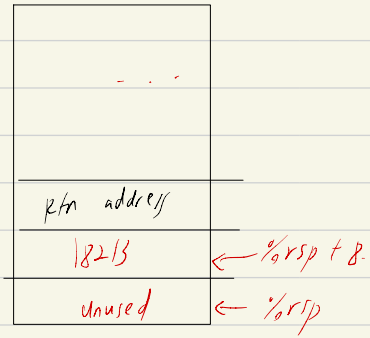
Example: Calling incr #3.

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

call_incr:

```
subq    $16     %rsp
movq    $15213  8(%rsp)
movl    $3000   %esi
leaq    8(%rsp) %rdi
call    incr
addq    8(%rsp) %rax
addq    $16     %rsp
ret
```

Stack structure



Register	Uses
%rdi	&v1
%rsi	3000
%rax	return value

Register saving conventions:

When procedure yoo calls who:

?
caller

↑
callee

Can register be used for temporary storage?

yoo:

```
...
movq    $15213  %rdx
call    who
addq    %rdx, %rax
...
ret
```

who:

```
...
subq    $18213, %rdx
...
ret
```

Contents of register %rdx overwritten by who.
This could be trouble → sth need be done.
need some coordination.

Register Saving Conventions

(I) When procedure yoo calls who:

yoo is the caller

who is the callee

(III) Conventions

(i) "caller saved"

Caller saves temporary values in its frame before the call.

(ii) "callee saved"

- Callee saves temporary values in its frame before using
- Callee restores them before returning to caller.

(I) %rax

(i) Return value

(ii) Also caller-saved

(iii) Can be modified by procedure

(II) %rdi, ..., %r9

(i) Arguments

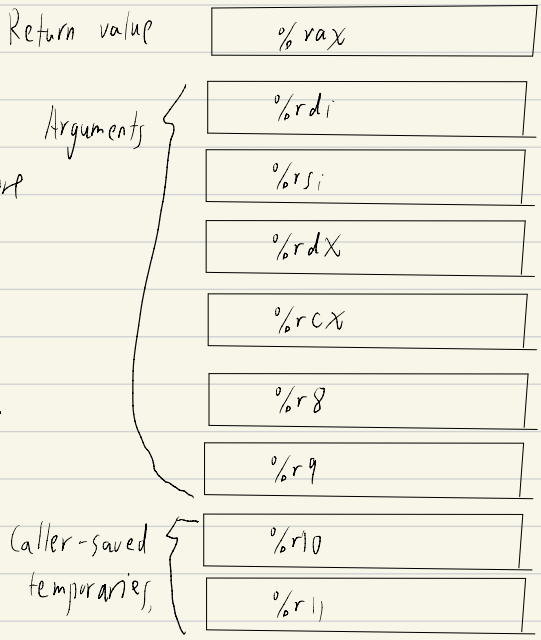
(ii) Also caller-saved

(iii) Can be modified by procedure

(III) %r10, %r11

(i) caller-saved

(ii) Can be modified by procedure



save temporary values that can be modified by any functions.

(I) %rbx, %r12, %r13, %r14

(i) callee-saved

(ii) callee must save & restore

(II) %rbp

(i) callee-saved

(ii) callee must save & restore

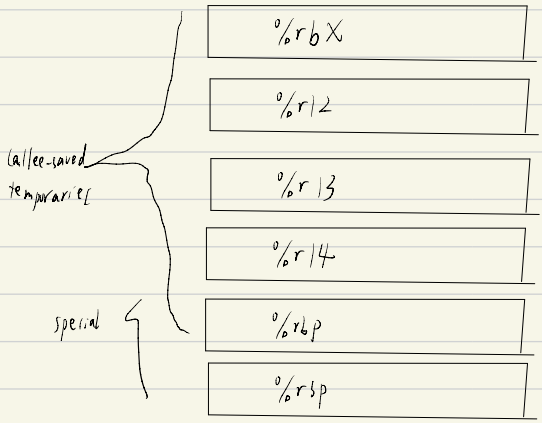
(iii) May be used as frame pointer

(iv) can mix & match.

(III) %rsp

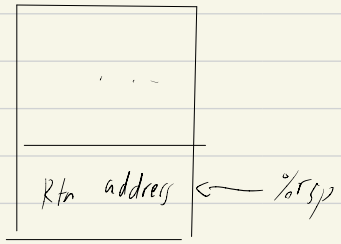
(i) Special form of callee save

(ii) Restored to original value upon exit from procedure



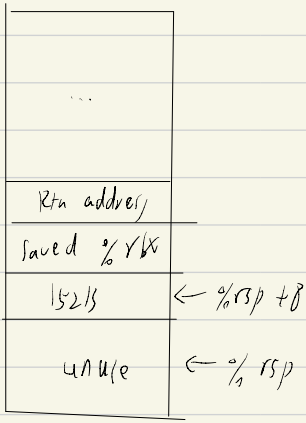
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x + v2;
}
```

Initial Stack Structure

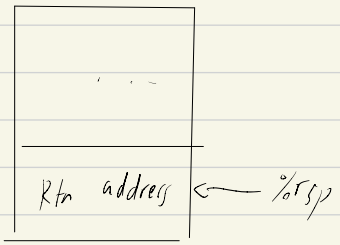



```
call_incr2:
    pushq   %rbx
    subq    $16,    %rsp    # allocate memory
    movq    %rdi,   %rbx
    movq    $15213, 8(%rsp)
    movl    $3000,  %esi
    leaq    8(%rsp), %rdi
    call    incr
    addq    %rbx,   %rax
    addq    $16,    %rsp
    popq    %rbx
    ret
```

Resulting Stack Structure



Pre-Return stack structure



Observations About Recursion

(I) Handled without special consideration

(i) Stack frames mean that each function call has private storage.

(ii) Saved registers and local variables.

(iii) Saved return pointer.

(II) register saving convention prevent one function call from corrupting another's data.

Unless the C code explicitly does so (see buffer overflow in Lecture 9)

(III) Stack discipline follows call / return pattern.

If P calls Q, then Q returns before P.

LIFO.

Also works for mutual recursion.

P calls Q; Q calls P.