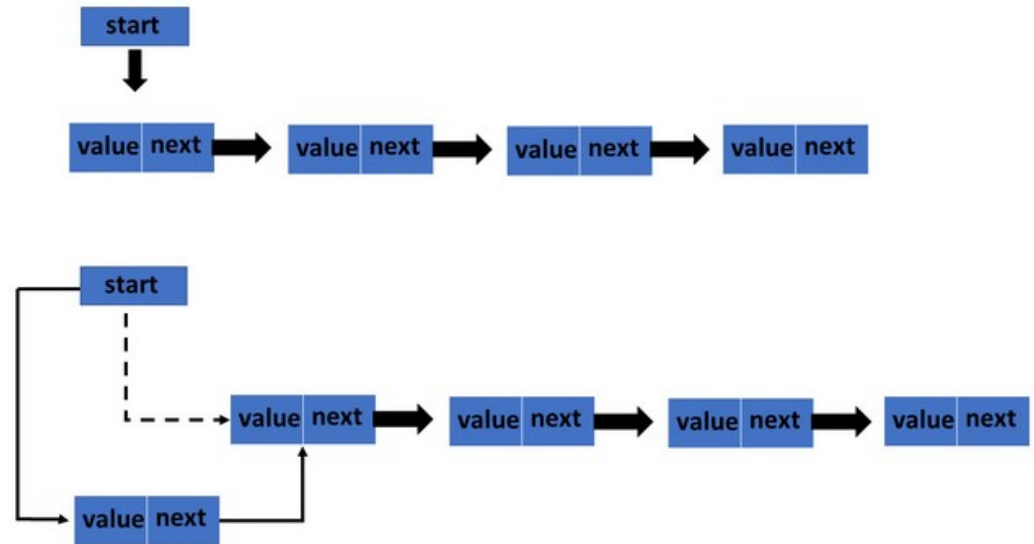


Stack Data Structure Pointers

- Variable that stores a memory address.
 - `int x = 10;`
 - `int *ptr = &10;`
 - `int y = *ptr;`

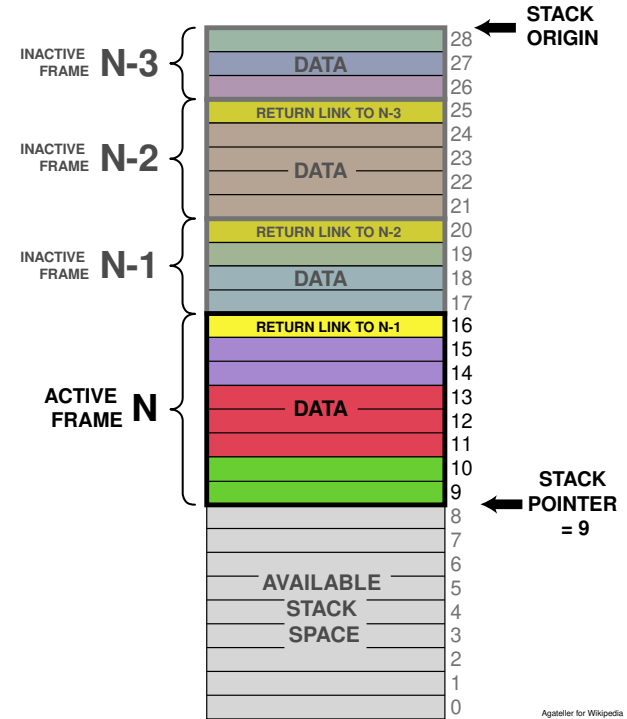
Stack Data Structure Linked List

- HEAD → NULL
- PUSH(ITEM1)
- HEAD → ITEM1 → NULL
- PUSH(ITEM2)
- HEAD → ITEM2 → ITEM1
- POP()
- HEAD → ITEM1



Stack Data Structure Review

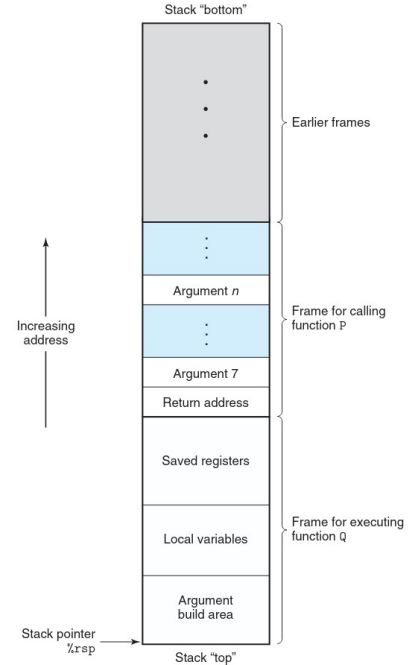
- Linked List
- Last inserted is at HEAD
- Oldest is at the bottom
- Push()
- Pop()



Run-Time Stack (ch 3.7.1)

- Last In, First Out (LIFO)
- Stack “grows” down
- Avoids stomping caller

Figure 3.25
General stack frame structure. The stack can be used for passing arguments, for storing return information, for saving registers, and for local storage. Portions may be omitted when not needed.



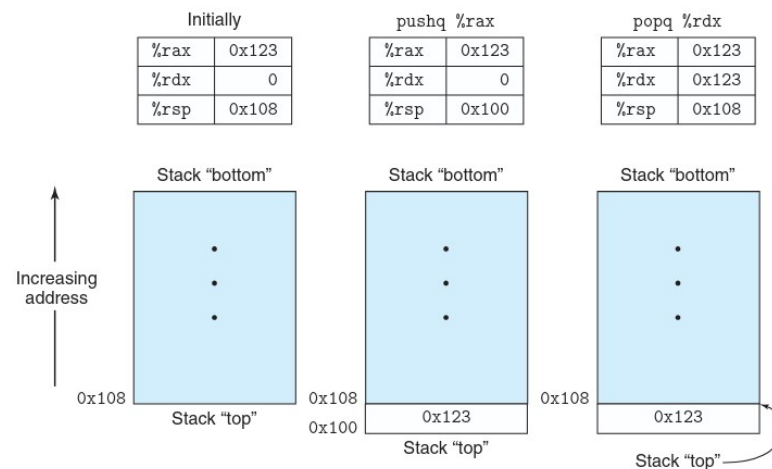
Run-Time Stack (ch 3.4.4)

- pushX
- popX

Instruction	Effect	Description
pushq <i>S</i>	$R[\%rsp] \leftarrow R[\%rsp] - 8;$ $M[R[\%rsp]] \leftarrow S$	Push quad word
popq <i>D</i>	$D \leftarrow M[R[\%rsp]];$ $R[\%rsp] \leftarrow R[\%rsp] + 8$	Pop quad word

Figure 3.8 Push and pop instructions.

C declaration	Intel data type	Assembly-code suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8



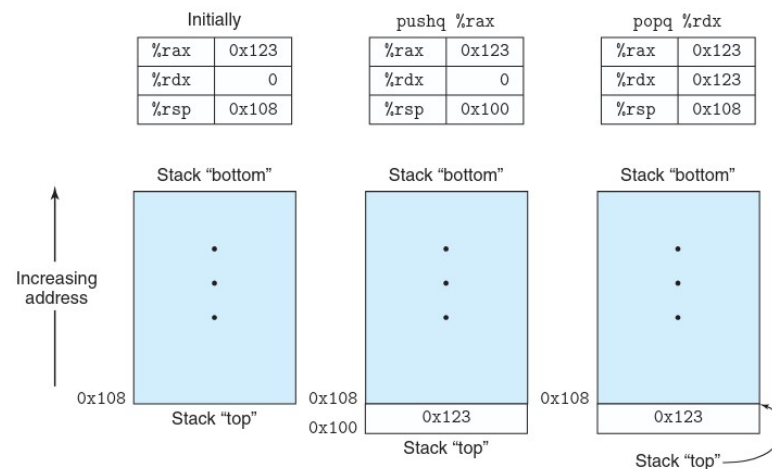
Run-Time Stack (ch 3.4.4)

- pushX
- popX

Instruction	Effect	Description
pushq <i>S</i>	$R[\%rsp] \leftarrow R[\%rsp] - 8;$ $M[R[\%rsp]] \leftarrow S$	Push quad word
popq <i>D</i>	$D \leftarrow M[R[\%rsp]];$ $R[\%rsp] \leftarrow R[\%rsp] + 8$	Pop quad word

Figure 3.8 Push and pop instructions.

C declaration	Intel data type	Assembly-code suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8



Registers (X86_64)

- You get 16 registers

Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	r_a	$R[r_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b, r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b, r_i)$	$M[Imm + R[r_b] + R[r_i]]$	Indexed
Memory	$(, r_i, s)$	$M[R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(, r_i, s)$	$M[Imm + R[r_i] \cdot s]$	Scaled indexed
Memory	(r_b, r_i, s)	$M[R[r_b] + R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(r_b, r_i, s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed

Figure 3.3 Operand forms. Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either 1, 2, 4, or 8.



ASM Maths

- inc/dec ++/--
- add/sub/imul
- xor/or/and
- AR shift
- Logical shift
- Variants for all sizes

Instruction		Effect	Description
leaq	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D+1$	Increment
DEC	D	$D \leftarrow D-1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \sim D$	Complement
ADD	S, D	$D \leftarrow D + S$	Add
SUB	S, D	$D \leftarrow D - S$	Subtract
IMUL	S, D	$D \leftarrow D * S$	Multiply
XOR	S, D	$D \leftarrow D \wedge S$	Exclusive-or
OR	S, D	$D \leftarrow D \vee S$	Or
AND	S, D	$D \leftarrow D \& S$	And
SAL	k, D	$D \leftarrow D \ll k$	Left shift
SHL	k, D	$D \leftarrow D \ll k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D \gg_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D \gg_L k$	Logical right shift