The x86 Architecture and its relationship with C

IEEE

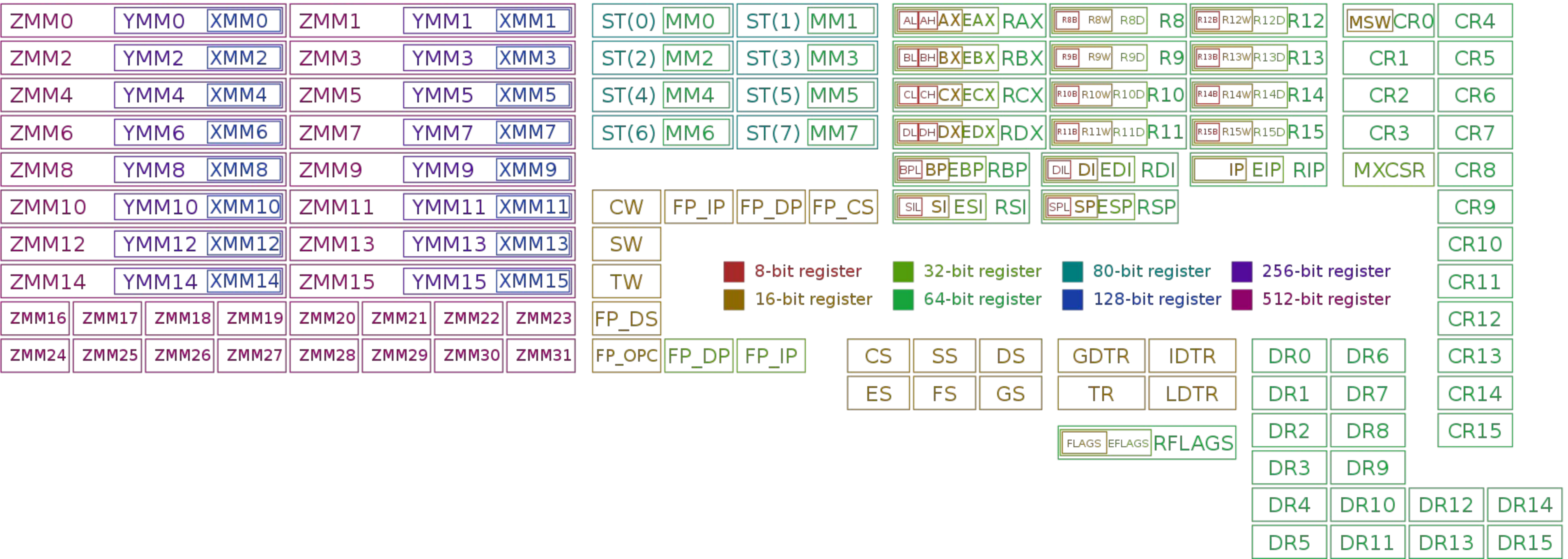
UAlbany Student Branch

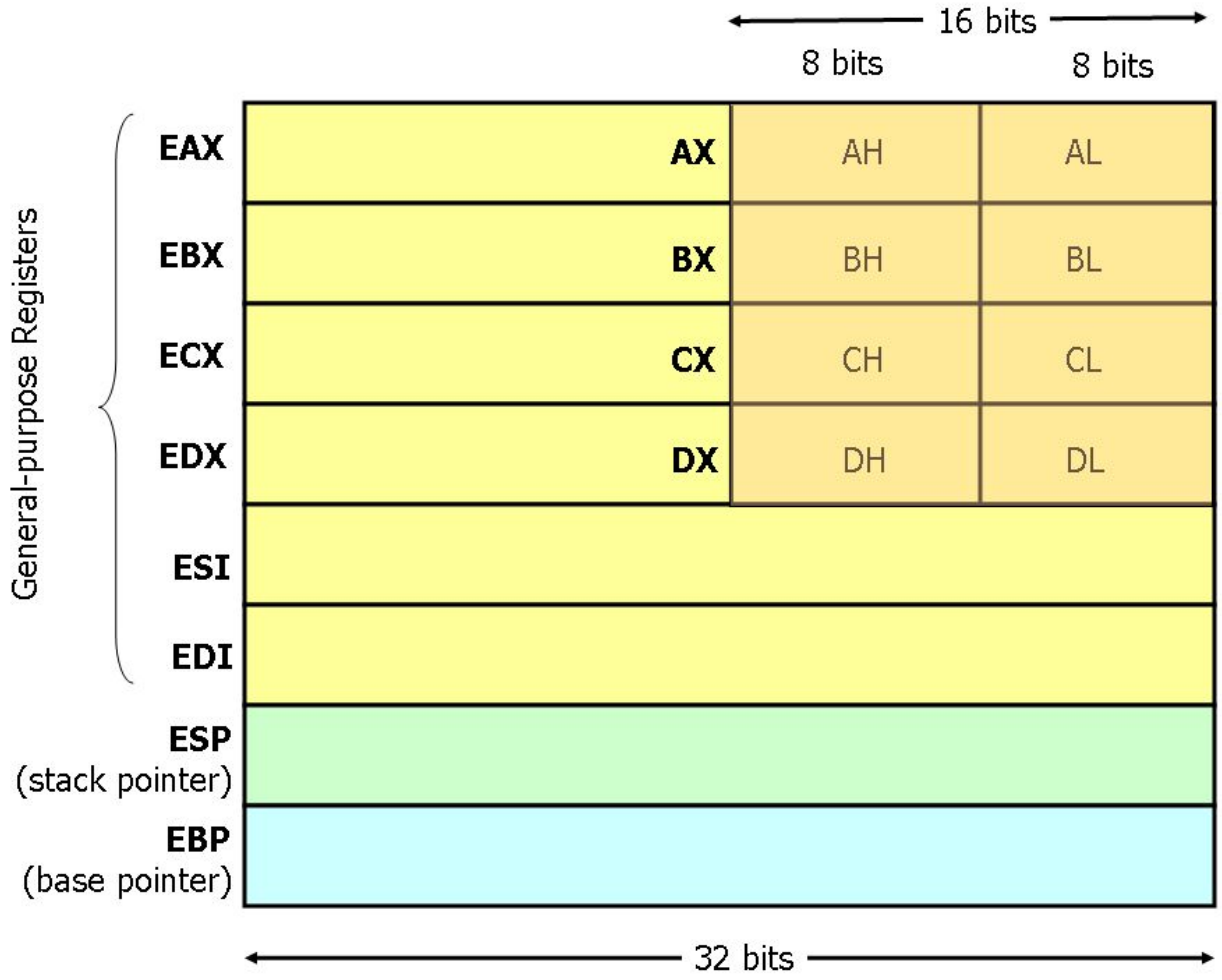
Glossary

Bits	Bytes	C type	Name
8	1	char	Byte
16	2	short	Word
32	4	int	Double Word
64	8	long	Quad Word

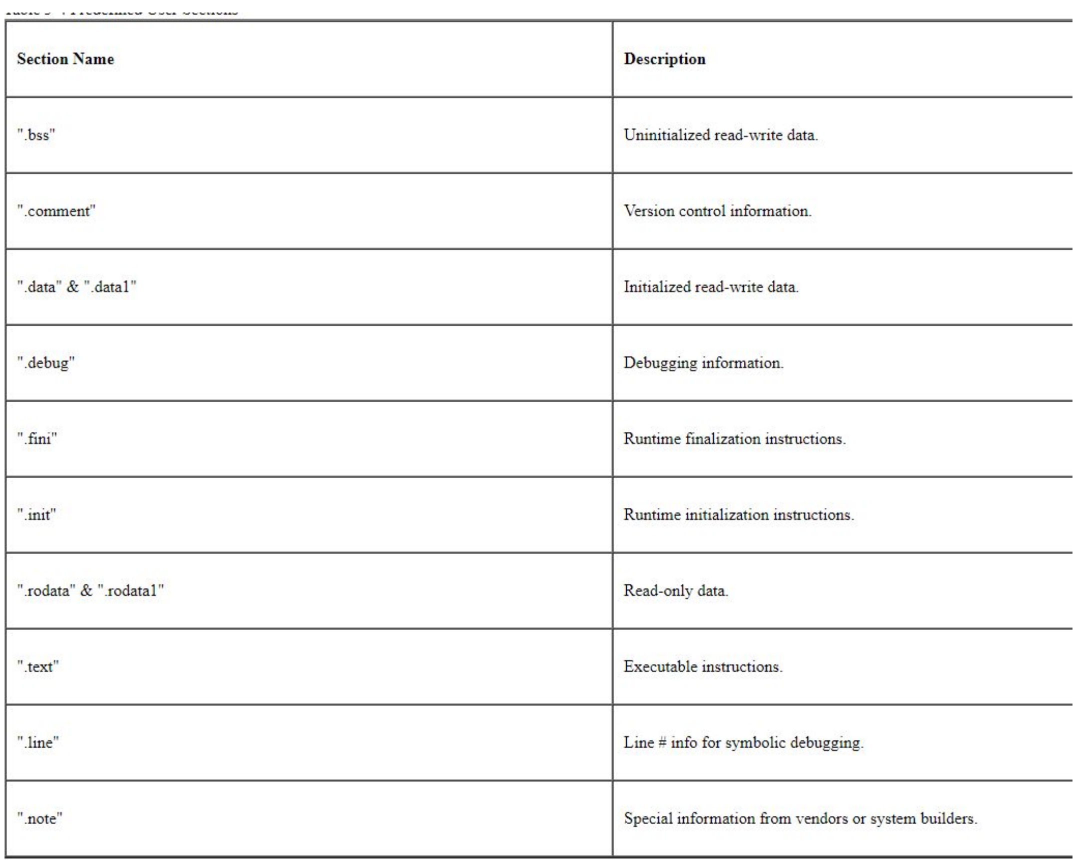
Registers

Registers are super fast memory locations on the processor that store values




16 bits
8 bits
8 bits
General-purpose Registers
EAX
AX
AH
AL
EBX
BX
BH
BL
ECX
CX
CH
CL
EDX
DX
DH
DL
ESI
EDI
ESP
(stack pointer)
EBP
(base pointer)
32 bits

Sections of an x86 program:



Section Name	Description
".bss"	Uninitialized read-write data.
".comment"	Version control information.
".data" & ".data1"	Initialized read-write data.
".debug"	Debugging information.
".fini"	Runtime finalization instructions.
".init"	Runtime initialization instructions.
".rodata" & ".rodata1"	Read-only data.
".text"	Executable instructions.
".line"	Line # info for symbolic debugging.
".note"	Special information from vendors or system builders.

https://docs.oracle.com/cd/E19455-01/806-3773/elf-3/index.html

Of these, we only care about

.bss .text .data

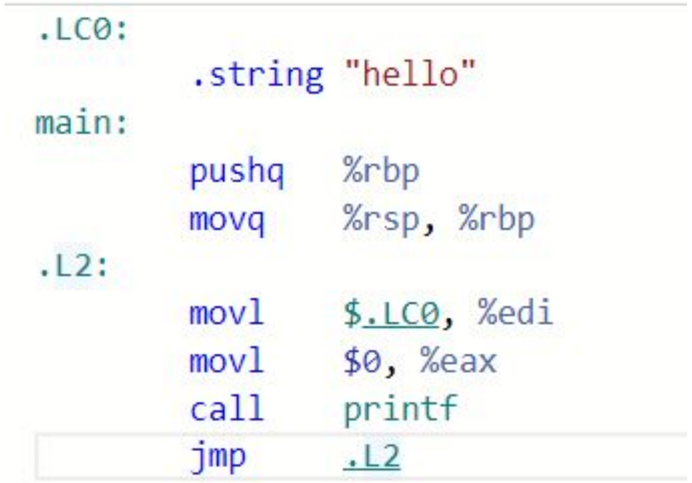
x86 Assembly structures

Type	Format	Use	Example
Instruction	*instructionName oprand1*, oprand2	Instructions for the processor to execute and their parameters	movl %eax, %ebx jmp $.L1
Directive	.*directiveName parameters*	Directives for the assembler,	.section .text .string “hello world!”
Label	*labelName*:	Represents a named location in memory	main:
Local Label	.*labelName*:	Represents a named location in memory in the scope of the parent label	.L1:

Labels

- Locations in memory, can be locations in the programs instructions or in data storage sections, like the stack.

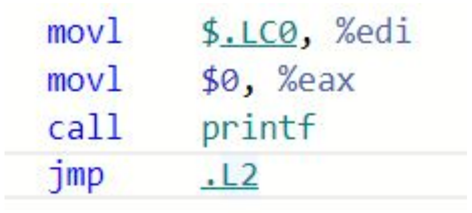
  Labels are primarily used for referencing stored data in the .bss and .data sections, as well as addresses in the .text section to jump to for program control.



```
.LC0:
        .string "hello"
main:
        pushq   %rbp
        movq    %rsp, %rbp
.L2:
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        jmp     .L2
```

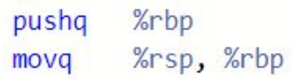
Instruction format

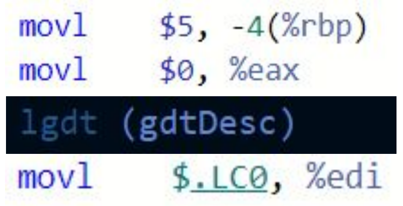
Instruction operands

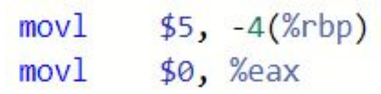


```
movl    $.LC0, %edi
movl    $0, %eax
call    printf
jmp     .L2
```

Types of operands

Operand Type	Format	Description	Example
Register	%regName	A register, always prefixed with % in gas syntax	 `pushq %rbp` `movq %rsp, %rbp`

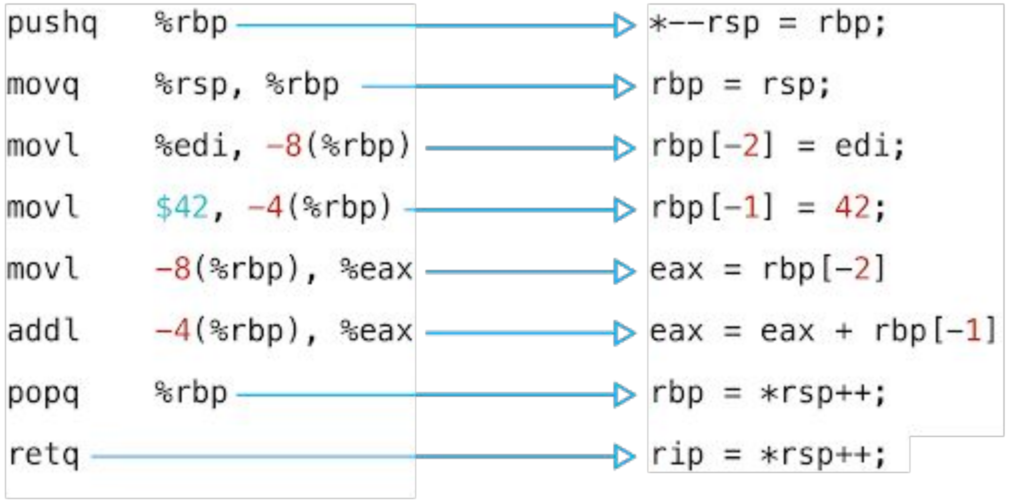
Memory Location	offset(%regName)
	offset(labelName)
		A memory location in RAM, Can either be a label or register with an offset, uses offset syntax or can be a label prefixed with $	 `movl $5, -4(%rbp)`
			`movl $0, %eax`
			`lgdt (gdtDesc)`
			`movl $.LC0, %edi`

Immediate	$constantValue	A constant value stored with the instruction itself, prefixed with $	 `movl $5, -4(%rbp)` `movl $0, %eax`

Basic instructions

Name	Syntax	Operation	Desc
move	movX r1, r2	r2 = r1	Move one value to another
add	addX r1, r2	r2 += r1	add two operands
subtract	subX r1, r2	r2 -= r1	subtract two operands
push	pushX r1	*rsp = r1 rsp += sizeof(X)	push the value of an operand to the stack
pop	popX r1	r1 = *rsp	Move the value of an operand off the stack
jump	jmp label	isp = &label	Jump control to a memory address
call	call label	push $ jump label	Jump to a memory address but set up for a return later
ret	ret	pop %reg Jump %reg	Return to the address left by the last call.

A sample program

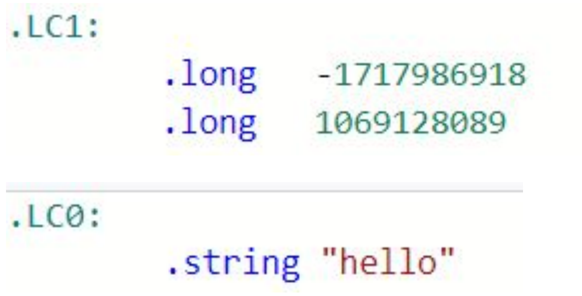


```
pushq   %rbp
movq    %rsp, %rbp
movl    %edi, -8(%rbp)
movl    $42, -4(%rbp)
movl    -8(%rbp), %eax
addl    -4(%rbp), %eax
popq    %rbp
retq
```

```
*--rsp = rbp;
rbp = rsp;
rbp[-2] = edi;
rbp[-1] = 42;
eax = rbp[-2]
eax = eax + rbp[-1]
rbp = *rsp++;
rip = *rsp++;
```

Directives

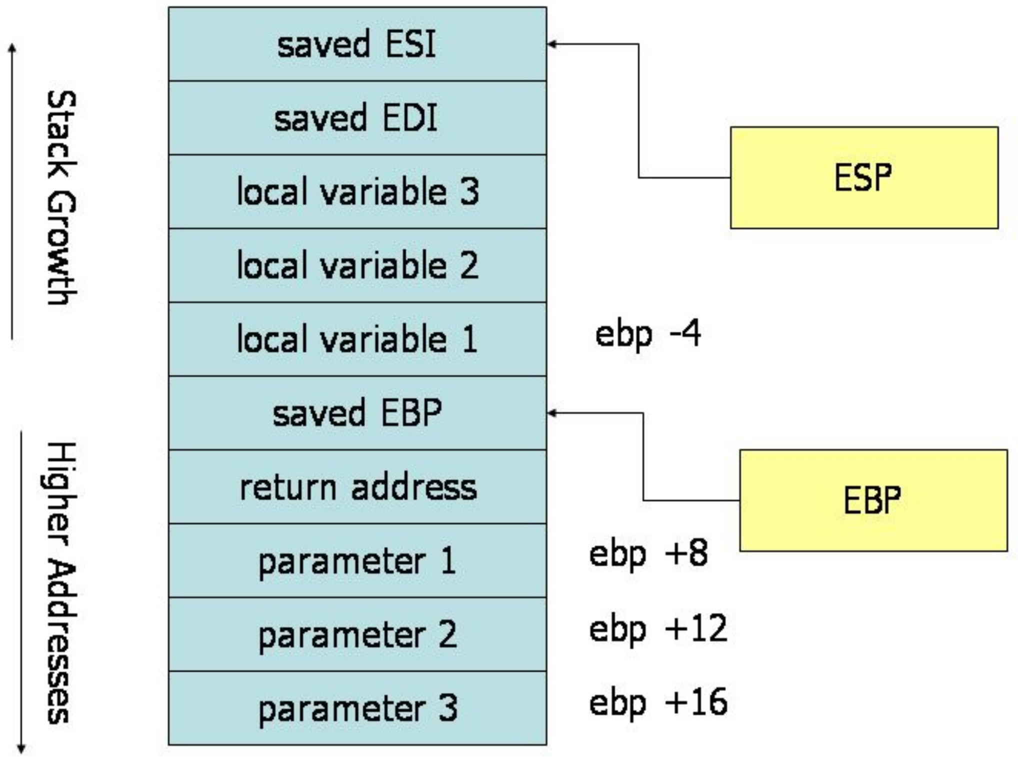
- Directives give the assembler special instructions about how to store the final machine code executable. Some common directives are those which store values into the data section.



```
.LC1:
        .long   -1717986918
        .long   1069128089
```

```
.LC0:
        .string "hello"
```

The Stack


Stack Growth
Higher Addresses
saved ESI
saved EDI
local variable 3
local variable 2
local variable 1
saved EBP
return address
parameter 1
parameter 2
parameter 3
ebp -4
ebp +8
ebp +12
ebp +16
ESP
EBP

Thank You for coming!