CS 261
Fall 2016

Mike Lam, Professor

Machine Code

Machine code

- We have studied multiple encodings of **information** (i.e., data)
  - Unsigned integers
  - Two's complement integers
  - ASCII / Unicode
  - Floating-point numbers
- We'll conclude by studying encodings of **instructions** (i.e., code)
  - Machine code
  - Assembly code
  - This will lead naturally to our next topic: CPU architectures

Instruction set architecture

- Every CPU has a set of instructions that it supports
  - Each instruction has a corresponding opcode
  - Most instructions also require parameters
    - Register numbers, memory addresses, immediate values
- Every CPU also maintain state information
  - Program counter: address of next instruction
  - Register file: quick-access memory locations
  - Condition and flag registers: status information
  - Vector registers: multiple data values

RISC vs. CISC

- RISC: Reduced Instruction Set Computing
  - Small, highly optimized set of instructions
  - Often requires load/store instructions to access memory
  - Often uses fixed-size instruction encoding
  - Examples: MIPS, DEC, SPARC, Power, ARM
- CISC: Complex Instruction Set Computing
  - Larger, more powerful set of instructions
  - Many instructions perform multiple actions
    - E.g., load-and-add or fused-multiply-and-add
  - Usually requires variable-sized instruction encoding
  - Examples: PDP-11, VAX, IA32, x86-64

Assembly code

- Machine code is made for machines
  - Very tedious for humans to read
- Assembly code: human-readable encoding of machine code
  - One instruction per line
  - Mnemonic for each opcode (e.g., “`add`”, “`jmp`”, “`halt`”)
  - Names for registers (e.g., “`%eax`”, “`%rax`”, “`%rbp`”)
  - Hex encoding of addresses and immediate values

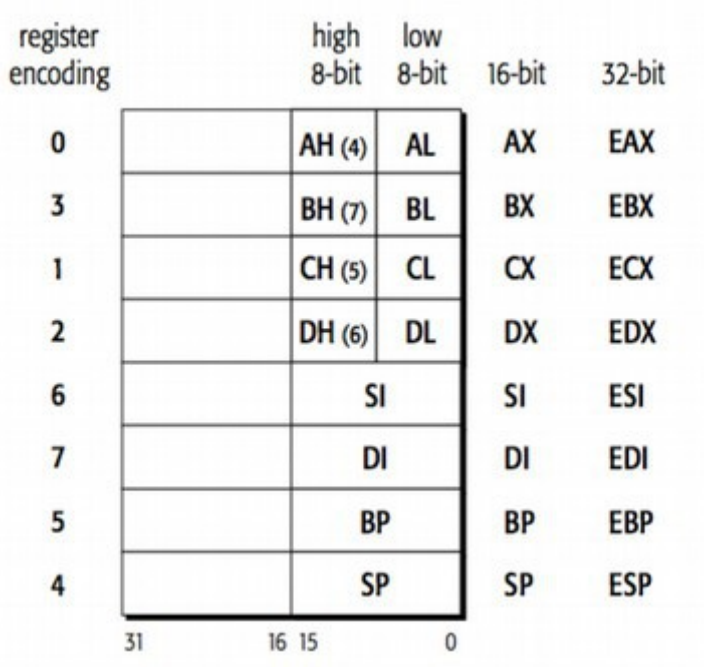
```
55                      push   %rbp
89 fa                   mov    %edi,%edx
88 45 f8                mov    %al,-0x8(%rbp)
01 d0                   add    %edx,%eax
5d                      pop    %rbp
c3                      retq
```

Examples

```
55      push %rbp

5d      pop %rbp

c3      retq
```



register encoding	high 8-bit	low 8-bit	16-bit	32-bit
0	AH (4)	AL	AX	EAX
3	BH (7)	BL	BX	EBX
1	CH (5)	CL	CX	ECX
2	DH (6)	DL	DX	EDX
6	SI		SI	ESI
7	DI		DI	EDI
5	BP		BP	EBP
4	SP		SP	ESP

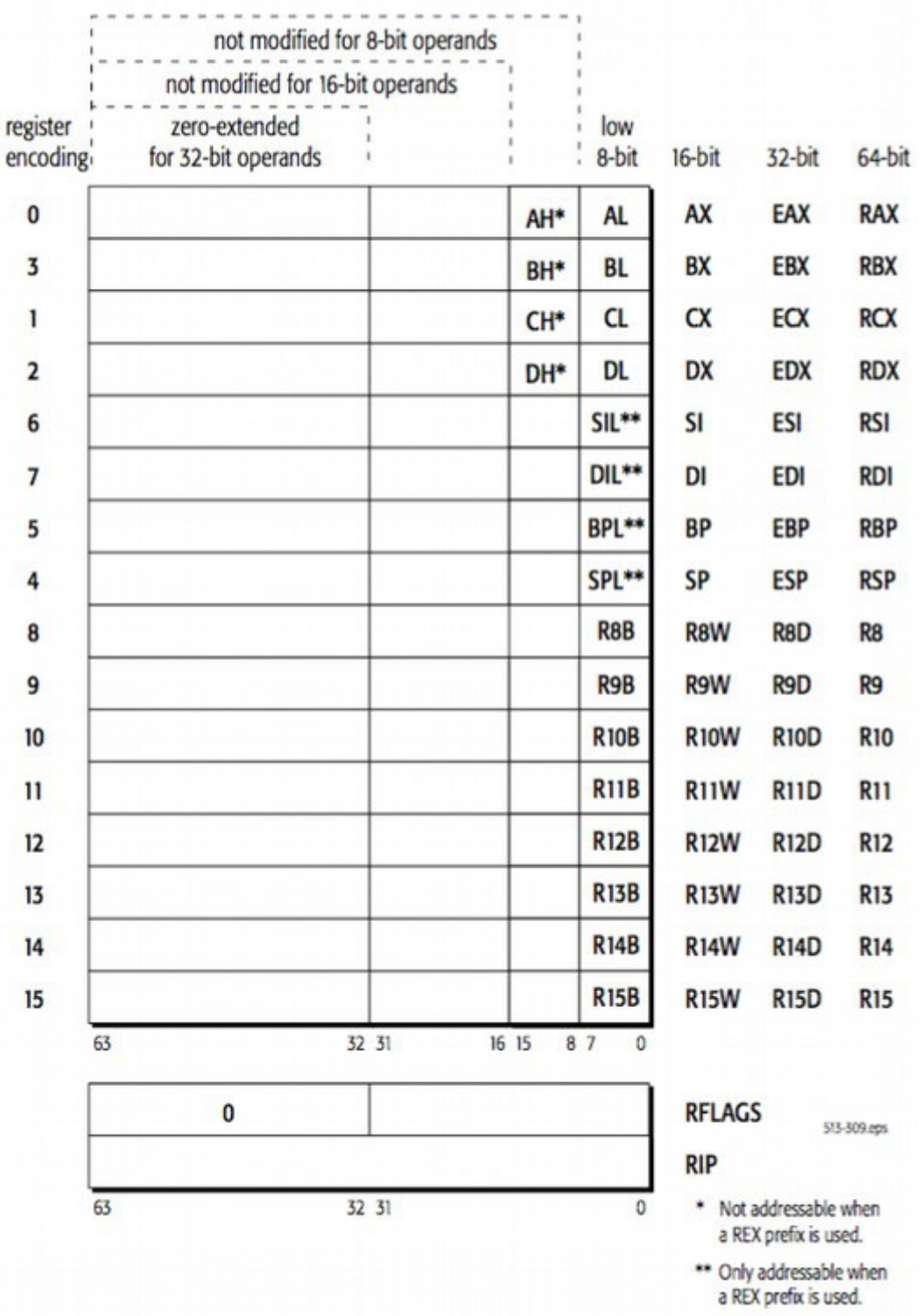
31 16 15 0

From the AMD64 manual (vol 3):

Mnemonic	Opcode	Description
PUSH *reg64*	50 +*rq*	Push the contents of a 64-bit register onto the stack.
POP *reg64*	58 +*rq*	Pop the top of the stack into a 64-bit register.
RET	C3	Near return to the calling procedure.

Registers

- General-purpose
  - AX: accumulator
  - BX: base
  - CX: counter
  - DX: address
  - SI: source index
  - DI: dest index
- Special
  - **BP: base pointer**
  - **SP: stack pointer**
  - **IP: instruction pointer**
  - **FLAGS: status info**



Tools

- Assembler
  - Converts assembly code into machine code
  - On stu: "`as`" (usually run via "`gcc`" driver)
- Disassembler
  - Extracts information and assembly code from machine code files
  - On stu: "`readelf`" and "`objdump`"
- Debugger
  - Step through the execution of machine code instructions
  - On stu: "`gdb`"