Introduction to IA-32

IA-32

- Assembly Language
  - 32-bit Intel
  - Most common personal computer architecture
  - Backwards compatible for IA-64
- Other Names
  - x86, x86-32, i386

History of IA-32

- History
  - Derives from Intel 16-bit architecture
  - First implemented on Intel's 80386 in 1985
  - Forked into 64-bit implementations
    - Intel's IA-64 in 1999
    - AMD's AMD64 in 2000

Reference Manuals

- Intel Developer's Manuals
  - Documentation Changes
  - Volume 1: Basic Architecture
  - Volume 2A: Instruction Set Reference A–M
  - Volume 2B: Instruction Set Reference N–Z
  - Volume 3A: System Programming Guide
  - Volume 3B: System Programming Guide

http://www.intel.com/products/processor/manuals/

Assembly Notation

- AT&T
  - Source precedes destination
  - Used commonly in old GNU tools (gcc, gdb, …)
  - Example:

```
mov $4, %eax   // GP register assignment
mov $4, %(eax) // Memory assignment
```

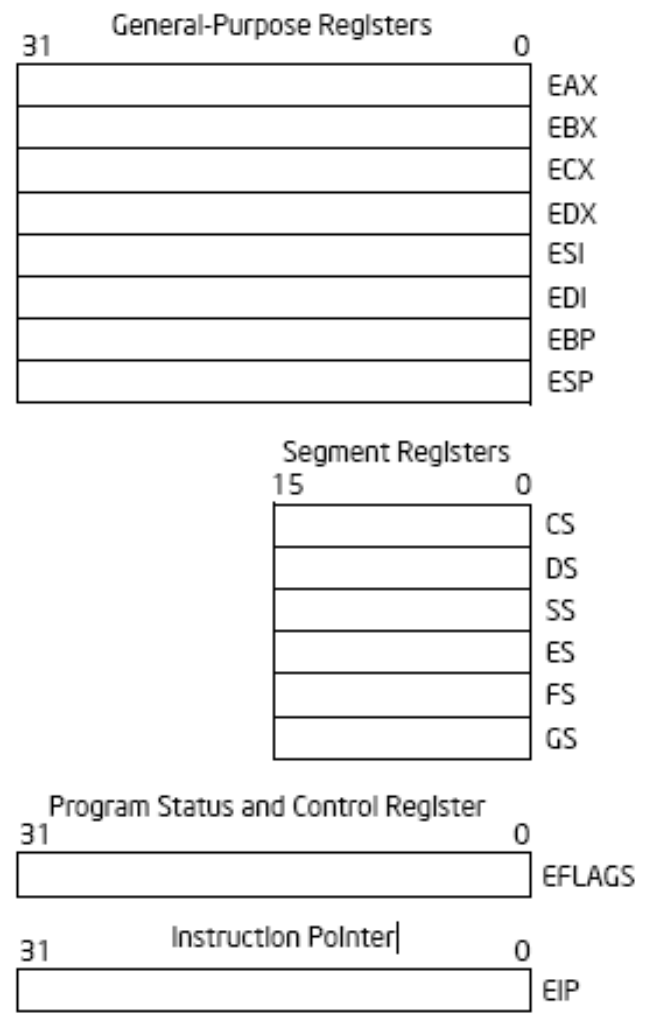
- Intel
  - Destination precedes source
  - Used elsewhere (MASM, NASM, …)
  - Example:

```
mov eax, 4   // GP register assignment
mov [eax], 4 // Memory assignment
```

Registers

- Processor Memory
  - Act as variables used by the processor
  - Are addressed directly by name in assembly code
  - Very efficient
    - Good alternative to RAM
  - Many flavors
    - Data registers
    - Address registers
    - Conditional registers
    - General purpose registers
    - Special purpose registers
    - …

IA-32 Registers



IA-32 Registers

- General Purpose Registers
  - EAX
    - General storage, accumulator, results
  - EBX
    - General storage, base, pointer for data in DS segment
  - ECX
    - General storage, counter
  - EDX
    - General storage, data, I/O pointer
  - ESI, EDI
    - General storage, pointer for memory copying operations
    - Source index, destination index

IA-32 Registers

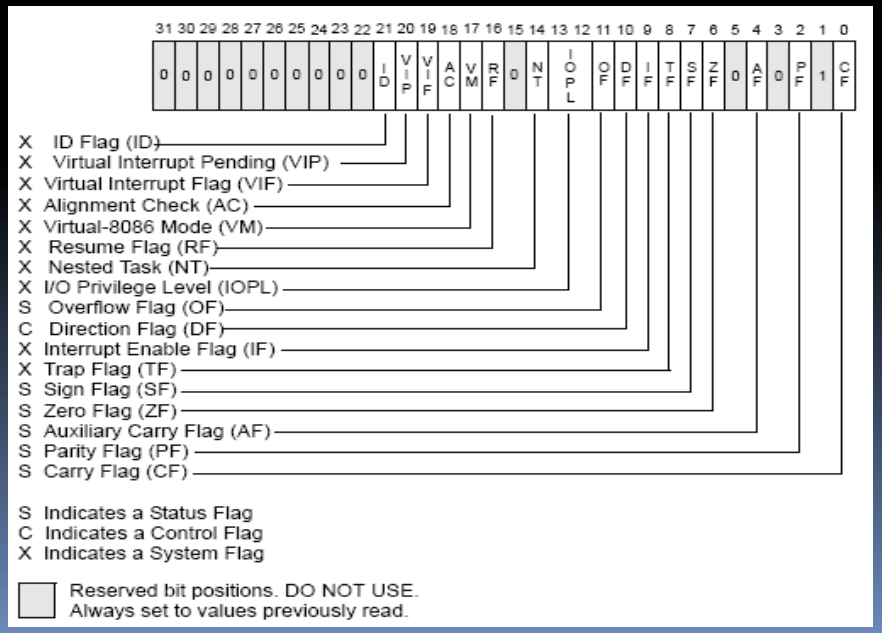
- General Purpose Registers
  - EBP
    - Stack "base pointer"
    - Current base of stack data
  - ESP
    - "Stack pointer"
    - Current location of the stack

IA-32 Registers

- Extended Instruction Pointer (EIP)
  - The program counter
  - Pointer to the next instruction
  - Altered by special instructions only
    - JMP, Jcc, CALL, RET, and IRET
  - Exploitation focuses on controlling the EIP

IA-32 Registers

- Status and Control (EFLAGS)
  - Processor info/modes, instruction status flags
  - Basis for conditional code execution

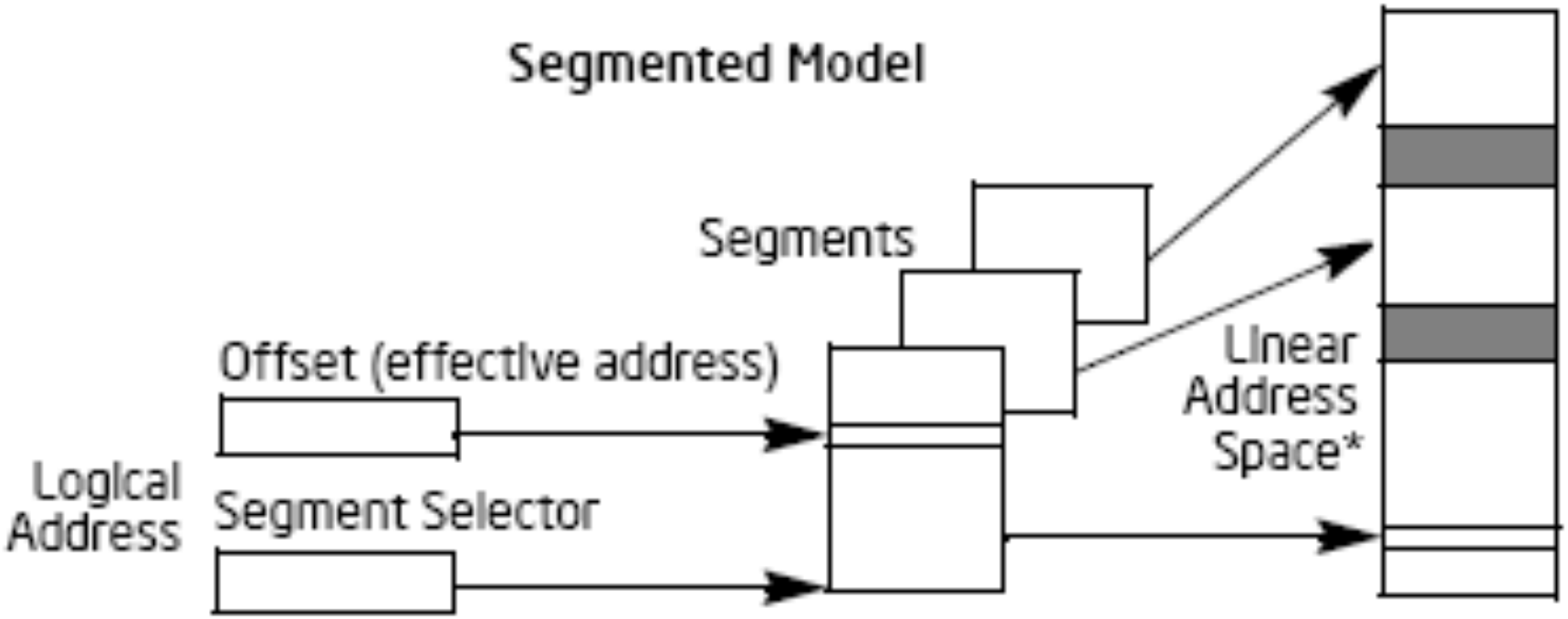


IA-32 Registers

- Important Flags
  - Carry flag (CF)
    - Set if an arithmetic operation generates a carry bit
  - Parity flag (PF)
    - Set if the least-significant byte of a result contains an even number of ones
  - Zero flag (ZF)
    - Set if the result is zero
  - Sign flag (SF)
    - Equal to the most significant bit of a result
  - Overflow flag (OF)
    - Set if integer overflows

Segmentation Memory Management Model

- Segmentation



IA-32 Registers

- Segment Registers
  - 16-bit memory segment selectors
  - CS
    - Code
    - Altered implicitly by calls, exceptions, etc.
  - DS
    - Data
  - SS
    - Stack
    - May be altered explicitly, allowing for multiple stacks

```
mov ss:[edx], eax  // Segment:[Offset]
```

IA-32 Registers

- Segment Registers
  - 16-bit memory segment selectors
  - ES
    - Data
  - FS
    - Data
  - GS
    - Data

IA-32 Registers

- Other Registers
  - FPU
    - ST0–ST7, status word, control word, tag word, …
  - MMX
    - MM0–MM7
    - XMM0–XMM7
  - Control registers
    - CR0, CR2, CR3, CR4
  - System table pointer registers
    - GDTR, LDTR, IDTR, task register
  - Debug registers
    - DR0, DR1, DR2, DR3, DR6, DR7

Alternate General Purpose Register Names

General-Purpose Registers

31 … 16	15 … 8	7 … 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Instruction Operands

- Instructions Operate on:
  - Registers
    - EIP cannot be an operand
      - Why? …What was EIP again?
  - Immediates
    - Literal, constant values

```
mov eax, 4
```

  - Memory addresses
    - Use other operands as pointers to address memory

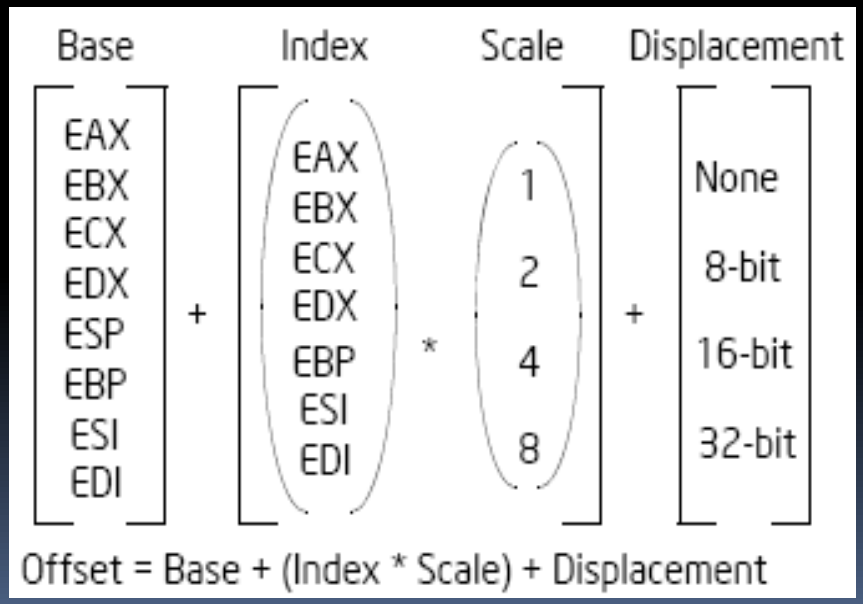
```
mov [eax], 4
```

Operand Addressing

- Instruction Addressing
  - Sources are addressed by:
    - Immediates
    - Pointers in registers
    - Pointers in memory locations
    - An I/O port
  - Destinations are addressed by:
    - Pointers in registers
    - Pointers in memory locations
    - An I/O port

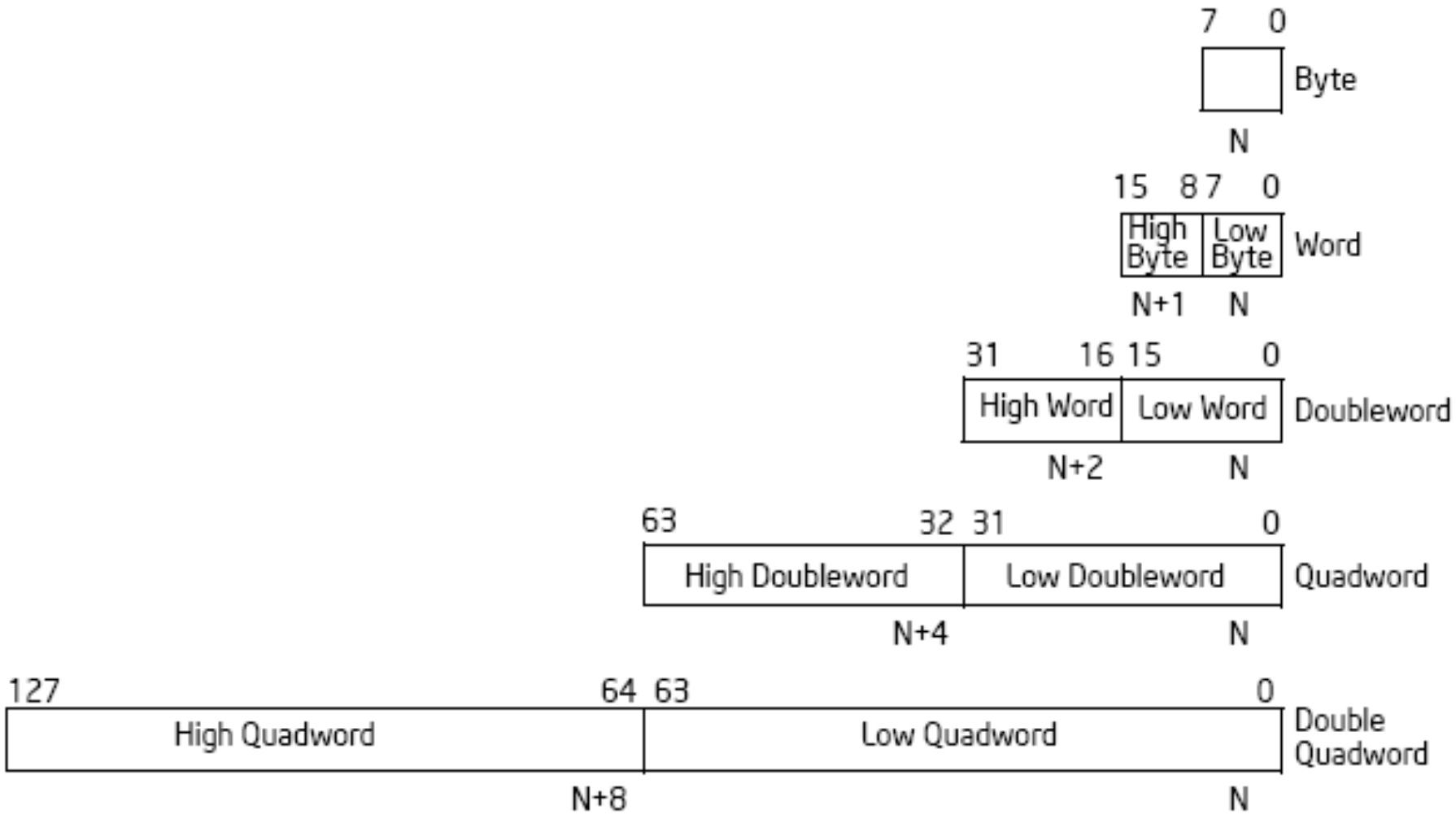
Operand Addressing

- Relative Offset Computation
  - Displacement
    - None, 8, 16, or 32-bits
  - Base
    - Value in GP register
  - Index
    - Value in GP register
  - Scale factor
    - 1, 2, 4, or 8
    - Multiplier for index




```
mov eax, [esi + ecx*4 + 4]
```

Data Types


7 0
Byte
N
15 8 7 0
High Byte
Low Byte
Word
N+1 N
31 16 15 0
High Word
Low Word
Doubleword
N+2 N
63 32 31 0
High Doubleword
Low Doubleword
Quadword
N+4 N
127 64 63 0
High Quadword
Low Quadword
Double Quadword
N+8 N

Common IA-32 Instructions

Move Instruction

- MOV
  - Moves a value from a source to a destination

```
mov eax, 4  // eax = 4
```

No Operation (NOP)

- NOP
  - Doesn't do anything
  - Handy placeholder
    - Also handy for shellcoding
  - Hex value
    - \x90

Arithmetic Instructions

- ADD, ADC
  - Add, add with carry

```
ADD eax, 1  // Equivalent to INC eax
```

- SUB, SUBB
  - Subtract, subtract with borrow
- MUL, IMUL
  - Multiply
- DIV, IDIV
  - Divide
- NEG
  - Two's-complement negate

Binary Logic Instructions

- AND, OR, NOT
  - And, or, not
- XOR
  - Xor trick (used by compilers and shellcoders)
    - Equivalent to "eax = eax ^ eax;" in C

```
xor eax, eax
```

Binary Operation Instructions

- SAL, SAR
  - Shift arithmetically left/right
- SHL, SHR
  - Shift logically left/right

Load Instructions

- LEA
  - May use relative or absolute address
  - Typically used to create an absolute address from relative offsets in a general purpose register
- LDS
  - Load pointer using DS
- LES
  - Load ES with pointer

Compare Instructions

- CMP (aka arithmetic compare)
  - Compares two numbers
    - Performs a subtraction (SRC1 – SRC2)
  - Sets CF, OF, SF, ZF, AF, and PF flags
- TEST (aka logical compare)
  - Compares two numbers
  - Sets SF, ZF, PF (also sets CF, OF to zero)

```
TEMP ← SRC1 AND SRC2;
SF ← MSB(TEMP);
IF TEMP = 0
THEN ZF ← 1;
ELSE ZF ← 0;
PF ← BitwiseXNOR(TEMP[0:7]);
CF ← 0;
OF ← 0;
```

Jump Instructions

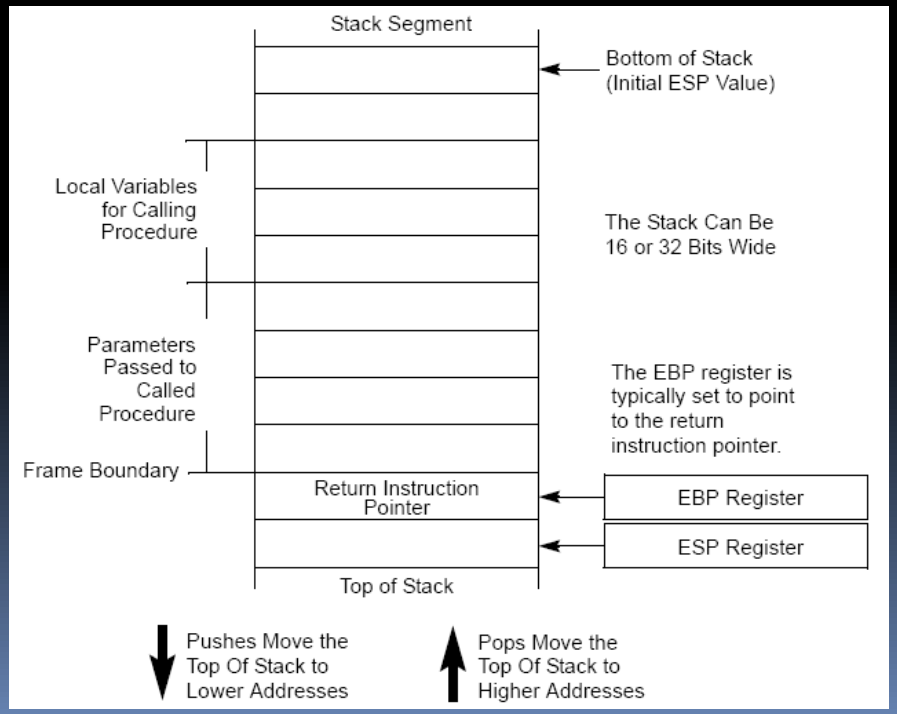
- JMP
  - Unconditional transfer of code execution
  - May use relative or absolute address

Conditional Jump Instructions

- Jcc
  - cc is called the conditional code
  - Conditional codes
    - JE/JZ (jump equal/zero, ZF = 1)
    - JNE/JNZ (jump not equal/not zero, ZF = 0)
    - JECXZ (jump ECX zero, ECX = 0)
    - JGE/JNL (jump greater, equal/not less, (SF xor OF) = 0)
    - …
  - JA, JAE, JB, JBE, JC, JCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ

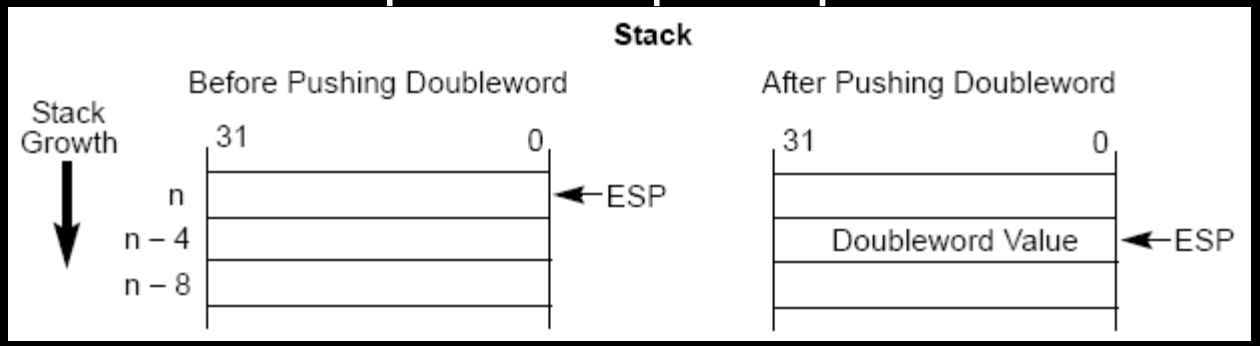
Stack

- LIFO Memory Structure
  - x86: stack grows downward (high to low addresses)

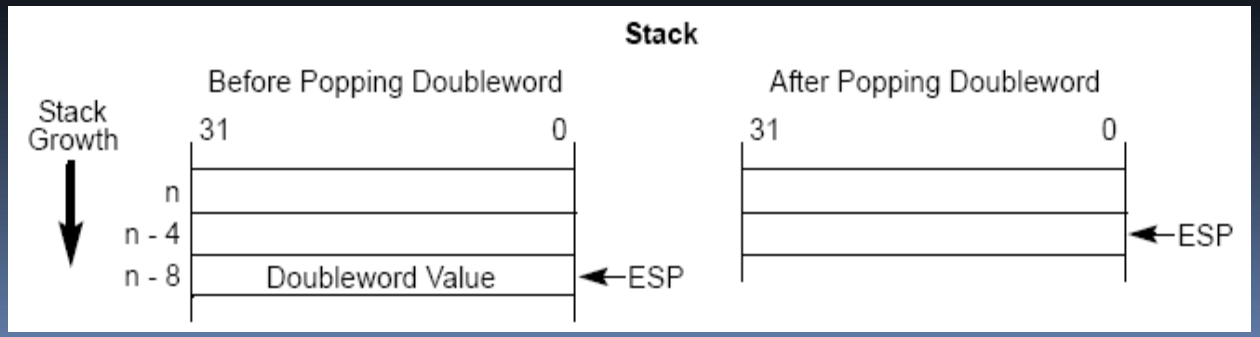


Stack Instructions

- PUSH
  - Decrement stack pointer, put operand at ESP



- POP
  - Load stack value, increment stack pointer



Stack Instructions

- PUSHA
  - Push all GP registers to the stack
- POPA
  - Pop data from stack into all GP registers
- ENTER
  - Enter stack frame

```
push ebp; mov ebp, esp
```

- LEAVE
  - Leave stack frame

```
mov esp, ebp; pop ebp
```

Near Call and Return Instructions

- Near Call/Return
  - Intrasegment call/return
  - Call or return to code in the same code segment
- Far Call/Return
  - Intersegment call/return
  - Call or return to code not in the same segment

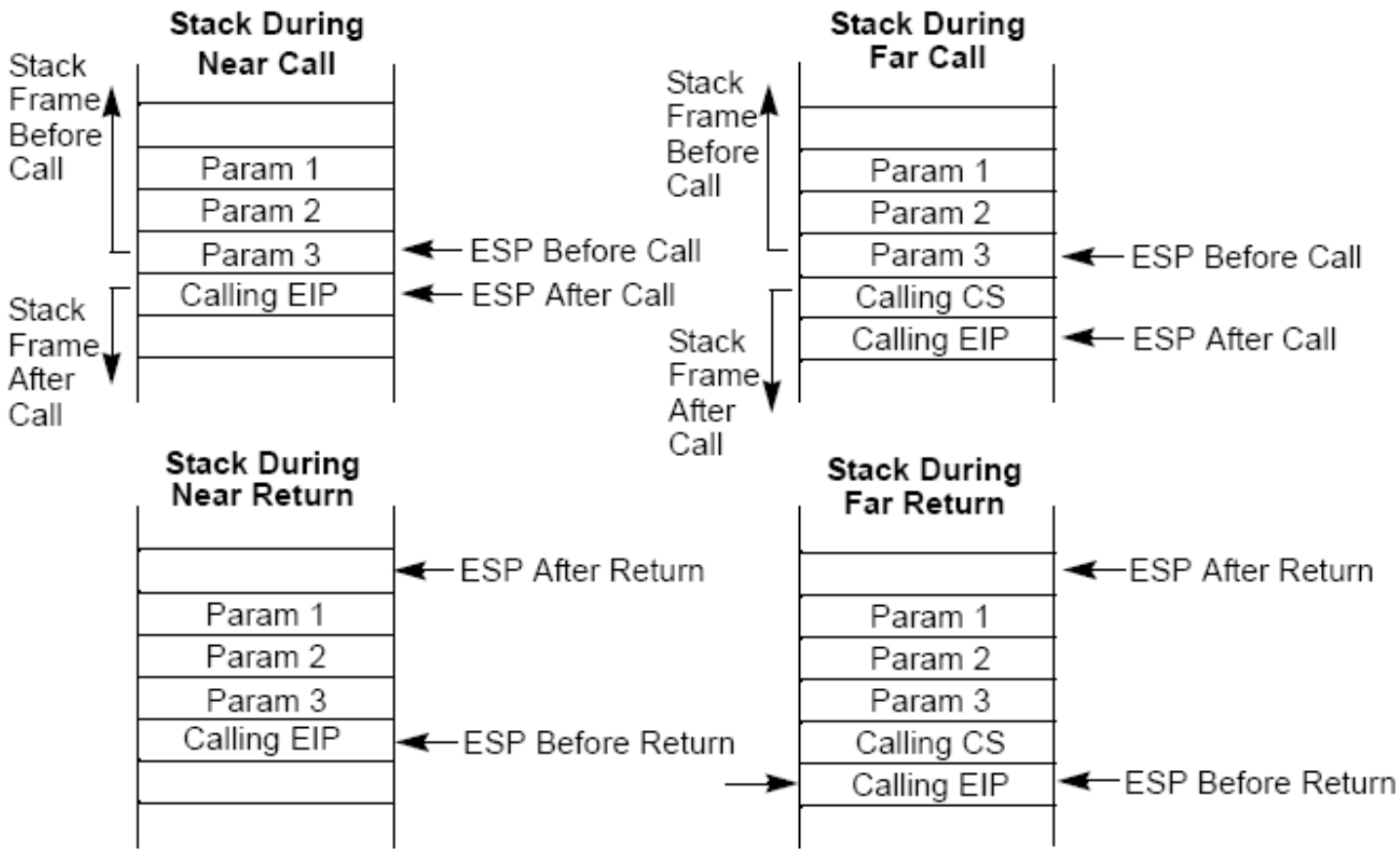
Near Call and Return Instructions

- Near Call (CALL)
  - Pushes the current EIP (the return address)
  - Loads the offset of the called procedure
- Near Return (denoted RET or RETN)
  - Pops the return address into EIP
  - If optional n argument, increment ESP by n
    - For clearing out parameters

Far Call and Return Instructions

- Far Call (CALL)
  - Pushes the current CS (the return code segment)
  - Pushes the current EIP (the return address)
  - Loads the CS, offset of the called procedure
- Far Return (denoted RET or RETF)
  - Pops the return address into EIP
  - Pops the return code segment
  - If optional n argument, increment ESP by n
    - For clearing out parameters

Calls and Returns



Note: On a near or far return, parameters are released from the stack based on the optional *n* operand in the RET *n* instruction.

Calls and Returns

```c
#include <stdio.h>
#include <stdlib.h>


void print_num(int, int, int);

void main(int argc, char *argv[]) {
    print_num(1, 2, 3);
}

void print_num(int i1, int i2, int i3) {
    printf("%n, %n, %n\n", i1,i2, i3);
}
```

```
00401031  . B8 CCCCCCCC    MOV EAX,CCCCCCCC
00401036  . F3:AB          REP STOS DWORD PTR ES:[EDI]
00401038  . 6A 03          PUSH 3
0040103A  . 6A 02          PUSH 2
0040103C  . 6A 01          PUSH 1
0040103E  . E8 C7FFFFFF    CALL functon_.0040100A
00401043  . 83C4 0C        ADD ESP,0C
00401046  . 5F             POP EDI
00401047  . 5E             POP ESI
00401048  . 5B             POP EBX

Registers (FPU)
EAX CCCCCCCC
ECX 00000000
EDX 00430D50
EBX 7FFD9000
ESP 0012FF28
EBP 0012FF80
ESI FFFFFFFF
EDI 0012FF80
EIP 0040103E functon_.0040103E

Address   Hex dump
00422000  00 00 00 00 00 00 00 00
00422008  00 00 00 00 00 00 00 00
00422010  00 00 00 00 00 00 00 00
00422018  00 00 00 00 00 00 00 00
00422020  00 00 00 00 00 00 00 00
00422028  00 00 00 00 00 00 00 00
00422030  00 00 00 00 00 00 00 00
00422038  00 00 00 00 00 00 00 00

0012FF28  00000001   Arg1 = 00000001
0012FF2C  00000002   Arg2 = 00000002
0012FF30  00000003   Arg3 = 00000003
0012FF34  7C910228   ntdll.7C910228
0012FF38  FFFFFFFF
0012FF3C  7FFD9000
0012FF40  CCCCCCCC
0012FF44  CCCCCCCC
0012FF48  CCCCCCCC
```

Calls and Returns

```c
#include <stdio.h>
#include <stdlib.h>


void print_num(int, int, int);

void main(int argc, char *argv[]) {
    print_num(1, 2, 3);
}

void print_num(int i1, int i2, int i3) {
    printf("%n, %n, %n\n", i1,i2, i3);
}
```

```
00401070  > 55          PUSH EBP
00401071  . 8BEC        MOV EBP,ESP
00401073  . 83EC 40     SUB ESP,40
00401076  . 53          PUSH EBX
00401077  . 56          PUSH ESI
00401078  . 57          PUSH EDI
00401079  . 8D7D C0     LEA EDI,DWORD PTR SS:[EBP-40
0040107C  . B9 10000000 MOV ECX,10
00401081  . B8 CCCCCCCC MOV EAX,CCCCCCCC
00401086  . F3:AB       REP STOS DWORD PTR ES:[EDI]
```

```
Registers (FPU)
EAX CCCCCCCC
ECX 00000000
EDX 00430D50
EBX 7FFD9000
ESP 0012FF24
EBP 0012FF80
ESI FFFFFFFF
EDI 0012FF80

EIP 00401070 functon_.print_num
```

```
Address   Hex dump
00422000  00 00 00 00 00 00 00 00
00422008  00 00 00 00 00 00 00 00
00422010  00 00 00 00 00 00 00 00
00422018  00 00 00 00 00 00 00 00
00422020  00 00 00 00 00 00 00 00
00422028  00 00 00 00 00 00 00 00
00422030  00 00 00 00 00 00 00 00
```

```
0012FF24  00401043  RETURN to functon_.main+23 from functon_.0040100A
0012FF28  00000001
0012FF2C  00000002
0012FF30  00000003
0012FF34  7C910228  ntdll.7C910228
0012FF38  FFFFFFFF
0012FF3C  7FFD9000
0012FF40  CCCCCCCC
```

String Operation Instructions

- INS, OUTS
  - Input/output string from/to a port
- MOVS, MOVSB, MOVSW, MOVSD
  - Moves data from one string to another
- LODS, LODSB, LODSW, LODSD
  - Loads data into a string (DS:[(E)SI] to (E)AX)
- STOS, STOSB, STOSW, STOSD
  - Store data in a string (ES:[(E)DI] with (E)AX)

String Operation Instructions

- CMPS, CMPSB, CMPSW, CMPSD
  - Compares strings in memory
- SCAS, SCASB, SCASW, SCASD
  - Compare a string (aka scan string)

Repeat String Operation Instructions

- REP, REPE, REPZ, REPNE, REPNZ
  - Repeats using the ECX register
  - REPxx
    - Where xx is a string operation instruction

Interrupt Instructions

- INT
  - Generate a software interrupt
  - INT 3h
    - Debugger breakpoint
    - Instruction hex value: \xCC or \xCD\x03
  - INT 80h
    - Unix system call
- RETI
  - Return from interrupt

Questions/Comments?

Some IA-32 Pictures from:

http://www.intel.com/products/processor/manuals/