Chapter 11 – Instruction Sets: Addressing Modes and Formats

Reading: Section 11.1 Addressing (pp. 387-394)

Registers

- General Purpose Registers (32 bits)
  - The primary accumulator register is called EAX. The return value from a function call is saved in the EAX register. Secondary accumulator registers are: EBX, ECX, EDX.
  - EBX is often used to hold the starting address of an array.
  - ECX is often used as a counter or index register for an array or a loop.
  - EDX is a general purpose register.
  - The EBP register is the stack frame pointer. It is used to facilitate calling and returning from functions.
  - ESI and EDI are general purpose registers. If a variable is to have register storage class, it is often stored in either ESI or EDI. A few instructions use ESI and EDI as pointers to source and destination addresses when copying a block of data. Most compilers preserve the value of ESI and EDI across function calls | not generally true of the accumulator registers.
  - The ESP register is the stack pointer. It is a pointer to the \top" of the stack.
  - The EFLAGS register is sometimes also called the status register. Several instructions either set or check individual bits in this register. For example, the sign flag (bit 7) and the zero flag (bit 6) are set by the compare (cmp) instruction and checked by all the conditional branching instructions.
  - The EIP register holds the instruction pointer or program counter (pc), which points to the next instruction in the text section of the currently running program.
- Segment Registers (16 bits) contain the base address of memory areas named segments. CS, SS, DS, ES, FS, GS

Segmented Memory

In real-address mode, the IA-32 processor family (80386-pentium 4) can access 1MB using 20 bit addresses in the range 0 to FFFFF. The problem: the 16-bit registers in the 8086 processor could not hold 20 bit addresses. The solution: segmented memory.

All of memory is divided into 64KB units called segments. Each segment begins at an address having zero for its last hex digit, so that digit can be omitted when representing the segment value. To reach a byte in a segment, an offset is used.

Addressing Modes
We have already discussed some addressing modes.

Q1: Can you name the addressing modes we have defined thus far?

Addressing Modes specify how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction

It provides the *means & ways* to *access* various operands in an assembly language program, and is completely architecture dependent.

There are three basic types of operands:

1. Immediate
    - Constant integer (8, 16, or 32 bits)
    - Constant value is stored within the instruction
2. Register
    - Name of a register is specified
    - Register number is encoded within the instruction
3. Memory
    - Reference to a location in memory
    - Memory address is encoded within the instruction, or
    - Register holds the address of a memory location

The most common addressing modes are:

1. Immediate Addressing

OPERAND = A

Example: ADD 5
Add 5 to contents of accumulator

Advantage: no memory reference.

Disadvantage: size of number is restricted to size of address field.

Examples:

MOV EAX,234H	MOV CX,2
MOV AL,34H	ADD AL,3
SUB CL,4	AND EAX,1

2. Direct Addressing

Address field contains address of operand

EA = A

Example: ADD A

- Add contents of cell A to accumulator
- Look in memory at address A for operand

Advantage: Single memory reference to access data
Disadvantage: Limited address space

3. Indirect Addressing

With direct addressing, length of address field is usually less than word length, thus limiting address range.

EA = (A)

Parentheses are to be interpreted as meaning contents of.

Example: ADD (A)

- Add contents of cell pointed to by contents of A to accumulator.

Advantage: for a word length of N, an address space of 2^N is now available.

Disadvantage: instruction execution requires two memory references to fetch operand:

- One to get its address
- A second to get its value

Indirect addressing can be nested, multilevel or cascaded.

EA = (...(A)...)

4. Register Addressing

Similar to direct addressing. Only difference is that address field refers to a register rather than a main memory address:

$$EA = R$$

Typically, an address field that references registers will have from 3 to 5 bits, so that a total of from 8 to 32 general-purpose registers can be referenced.

Advantages

- Only a small address field is needed in instruction.
- No memory references are required.

Disadvantage: address space is very limited.

```
MOV EAX,EBX      MOV CX,DX
MOV AH,AL        MOV AX,DS
ADD AL,CL        OR AX,DX
```

5. Register Indirect Addressing

Similar to indirect addressing. Only difference is that address field refers to a register rather than a main memory address:

$$EA = (R)$$

Operand is in memory cell pointed to by contents of register R.

Advantages: basically same for indirect addressing. Register indirect addressing has a large address space (2^N). Register indirect addressing uses one less memory reference than indirect addressing.

```
MOV AL,[BX]      MOV AX[EBX]
MOV [EDI],EAX    MOV [EAX],EDX
```

6. Displacement Addressing

Combines capabilities of direct addressing and register indirect addressing as

$$EA = A + (R)$$

Address field holds two values:

- A = base value
- R = register that holds displacement

```
MOV AL,[BX+SI]     MOV [BX+DI],AX
MOV [BP+SI],EAX    MOV AL,[BP+DI]
MOV WORD PTR [BX+SI],5
ADD AL,[BX+DI]
```

7. Stack Addressing

Associated with stack is a pointer whose value is address of top of stack.

EA = top of stack

Stack pointer is maintained in a register.

Machine instructions need not include a memory reference but implicitly operate on top of stack.

Example:ADD

- Pop top two items from stack and add

Write a program to copy 55H into RAM memory locations 40H to 45H using:

1. Direct addressing mode
2. Register indirect addressing mode without a loop
3. With a loop