18. Instruction Sets: Addressing Modes and Formats

Chapter 13, sections 13.1 and 13.2

x86 Real Mode
General Purpose Registers

- The primary accumulator register is called AX. The return value from a function call is saved in the AX register.
- Secondary accumulator registers are: BX, CX, DX.
  - BX is often used to hold the starting address of an array.
  - CX is often used as a counter or index register for an array or a loop.
  - DX is a general purpose register.
- The BP register is the stack frame pointer. It is used to facilitate calling and returning from functions.

x86 Real Mode
General Purpose Registers

- SI and DI are general purpose registers. A few instructions use SI and DI as pointers to source and destination addresses when copying a block of data (e.g. strings).
- The SP register is the stack pointer. It is a pointer to the “top" of the stack.
- The EFLAGS register is a 32 bit 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.

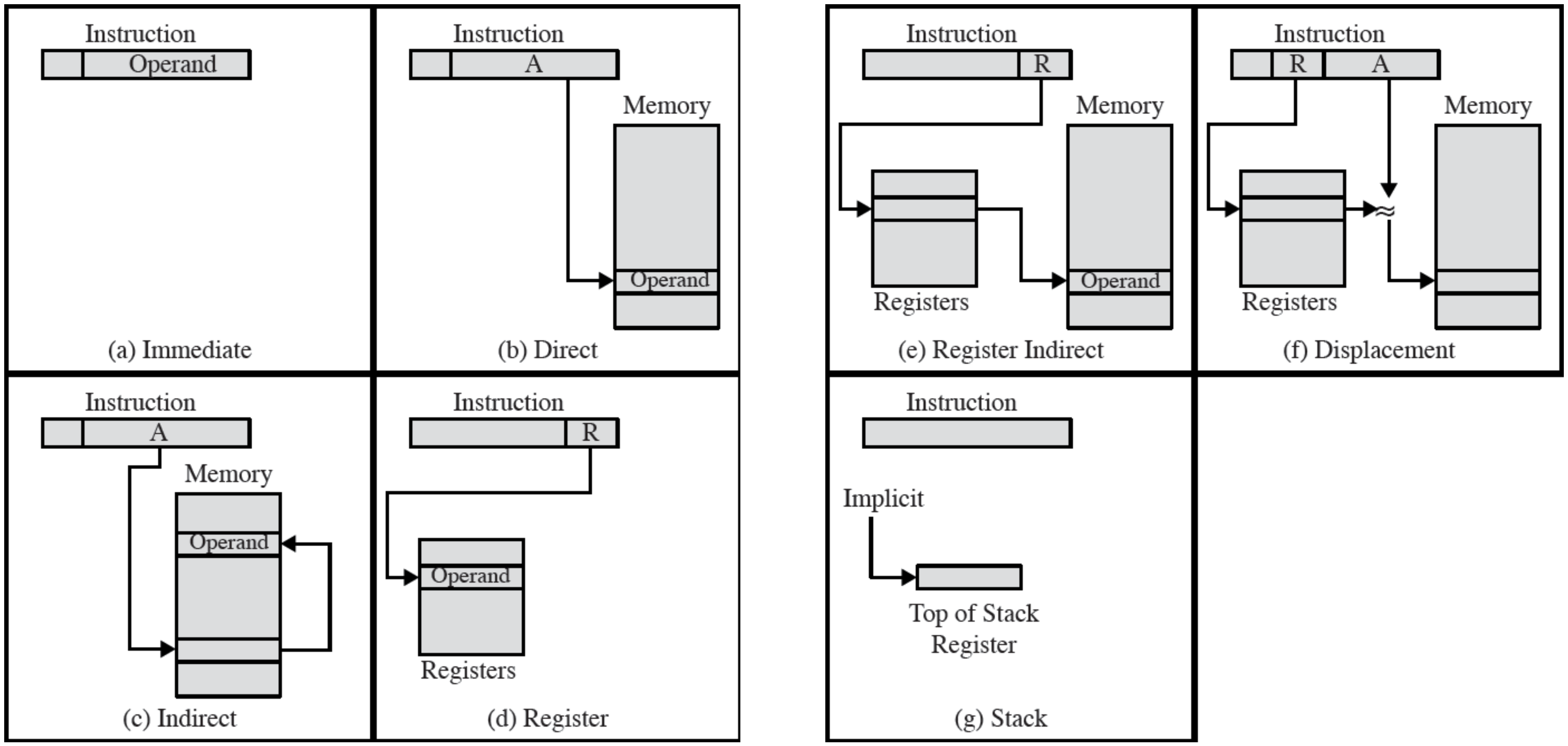
Addressing Modes

- 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
- Effective address: actual address of the location containing the referenced operand
- It provides the *means* & *ways* to *access* various operands in an assembly language program, and is completely architecture dependent

Basic Operand Types

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

Addressing Modes



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.
- x86 Examples:
  - MOV EAX,0x234 MOV CX,2
  - MOV AL,0x34 ADD AL,3
  - SUB CL,4 AND EAX,1

Direct Addressing
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
- x86 Examples
  - mov count, [var]

Indirect Addressing
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 the address
  - A second to get the value
- x86 examples - none

Register Addressing
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.
- Advantage:
  - Only a small address field is needed in instruction.
  - No memory references are required.
- Disadvantage
  - Address space is very limited.
- x86 Examples
  MOV EAX,EBX   MOV CX,DX

Register Indirect Addressing
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.

- x86 Example

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

Displacement Addressing
EA = A + (R)

- Address field holds two values:
  A = base value
  R = register that holds displacement

- x86 Example
  MOV AX, NAME[SI]

Stack Addressing
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, add, place the result on top of the stack