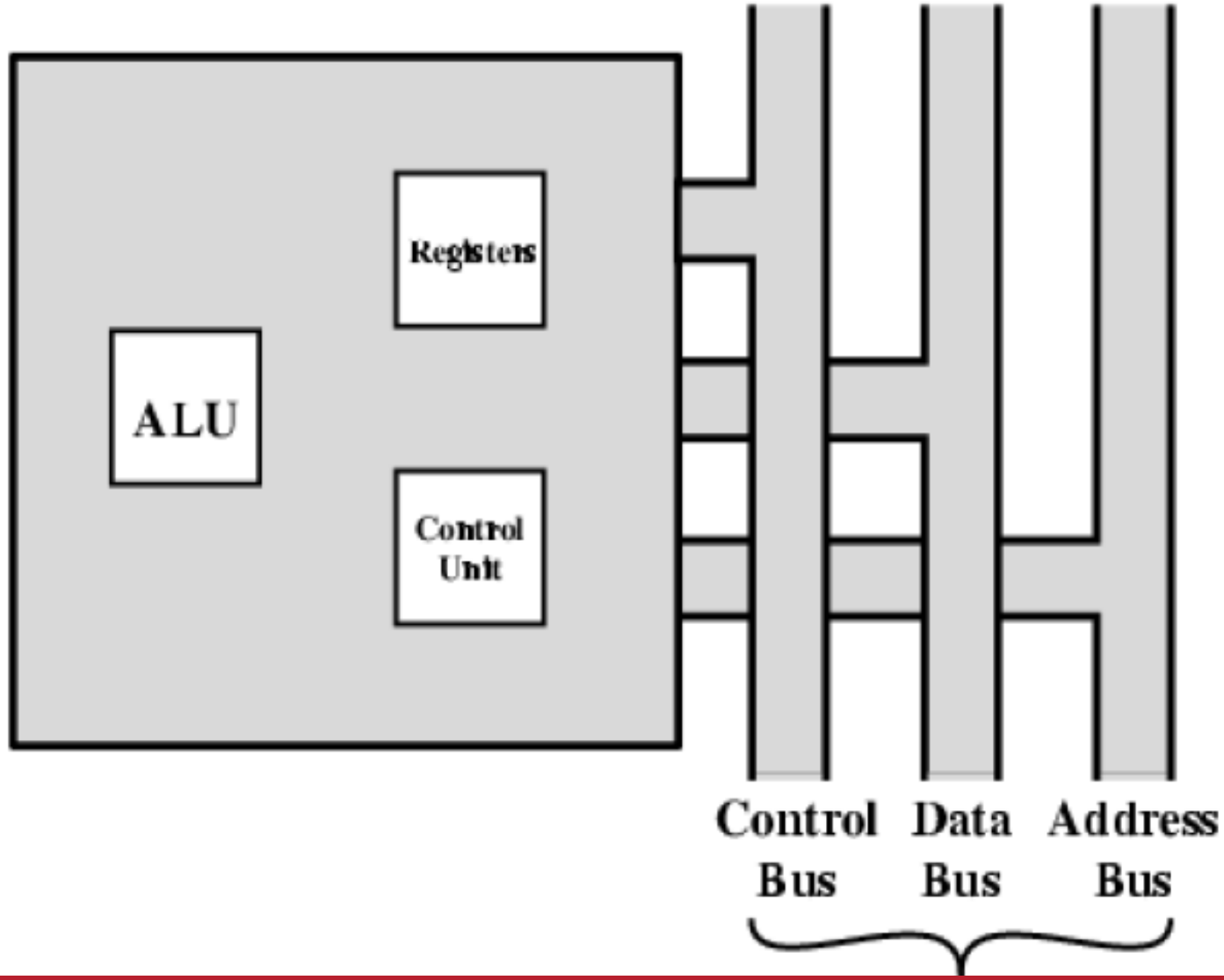
MODULE-3

Central Processing Unit	
3.1	CPU architecture, Register organization, Instruction formats and addressing modes(Intel processor).,Basic instruction cycle. Control unit Operation ,Micro operations : Fetch, Indirect, Interrupt , Execute cycle Control of the processor, Functioning of micro programmed control unit, Micro instruction Execution and Sequencing, Applications of Micro programming
3.2	RISC v/s CISC processors, RISC and CISC Architecture, RISC pipelining, Case study on SPARC

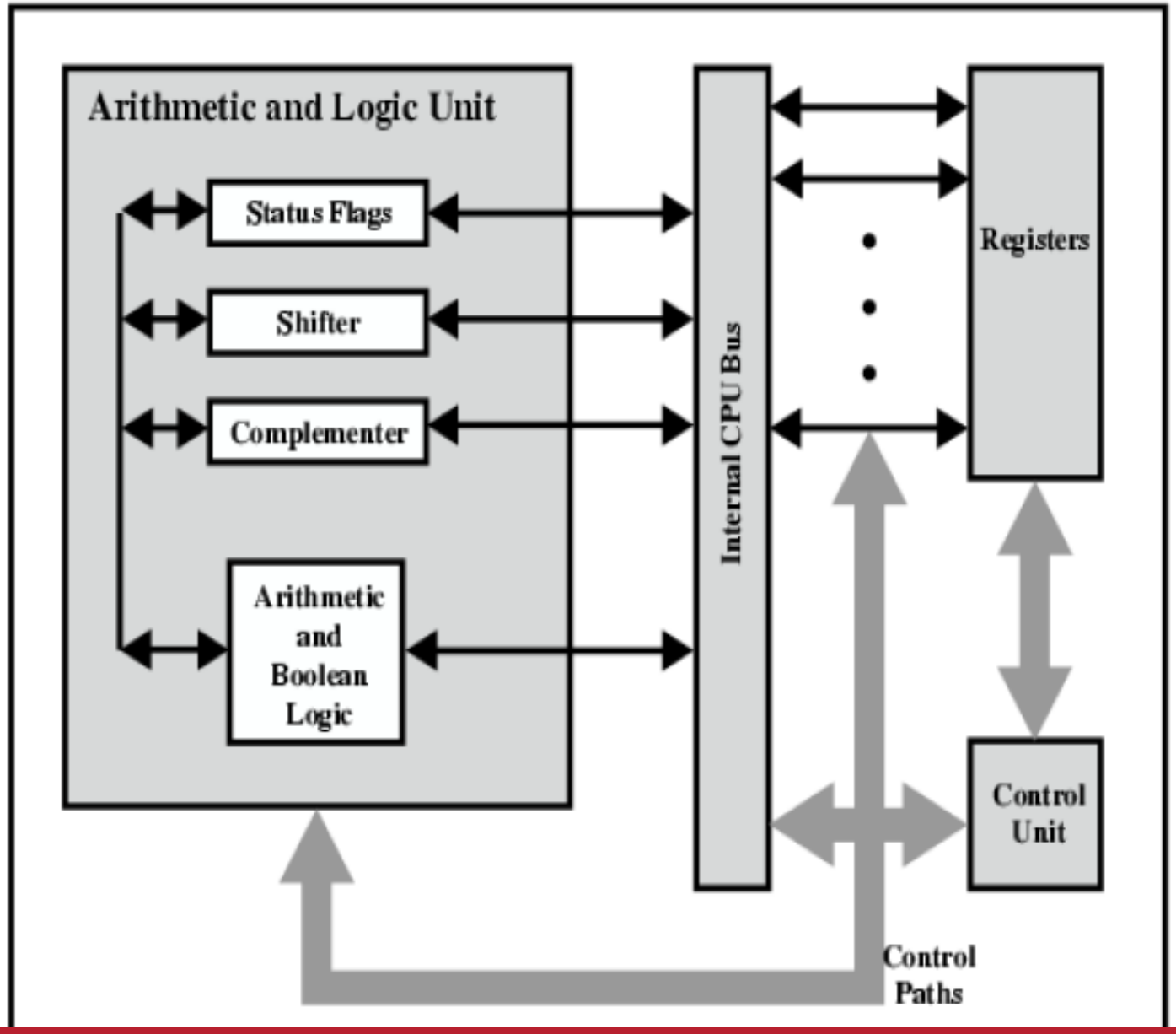
CPU Structure

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

CPU With Systems Bus



CPU Internal Structure



Registers

- CPU must have some working space (temporary storage)
- Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

Registers in the processor

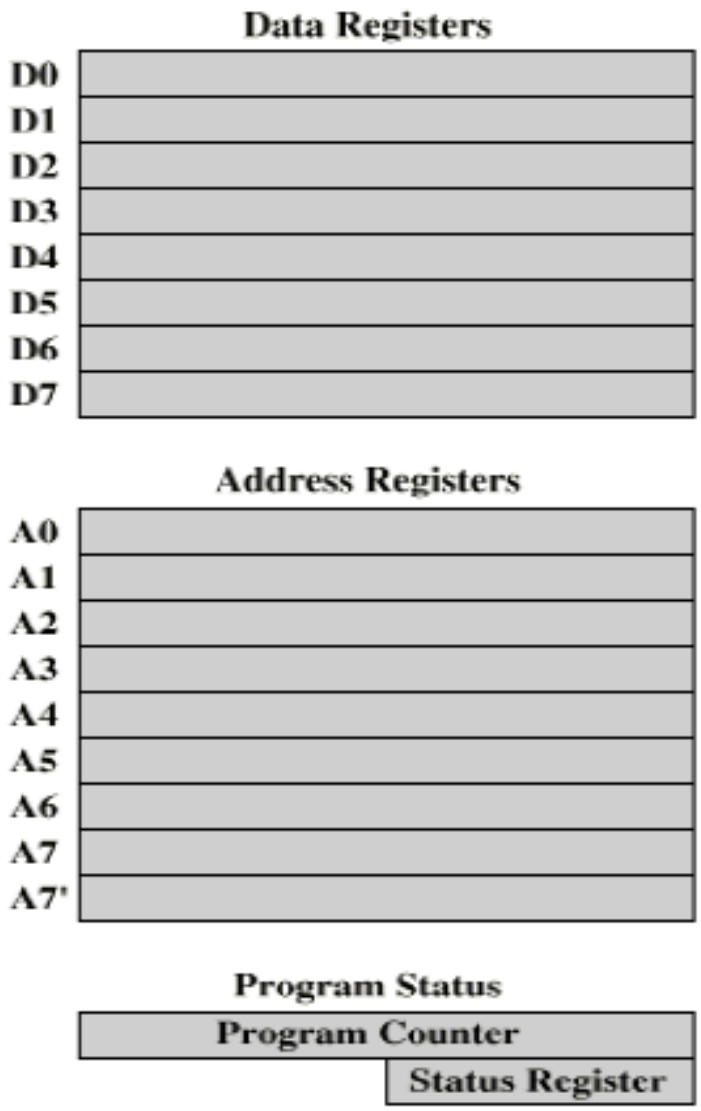
User –Visible Registers -Referenced by means of Assembly or machine language. Optimizing these can reduce reference to main memory.

- Control & Statue Registers- To Control the operation of processor, not visible to user.

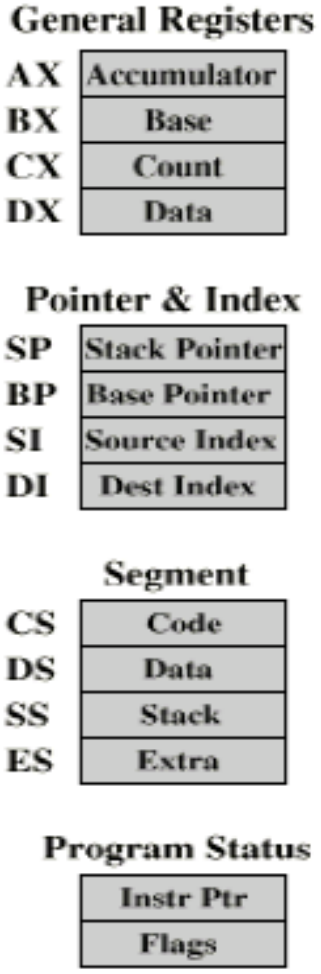
User Visible Registers

- General Purpose
- Data
- Address
- Condition Codes

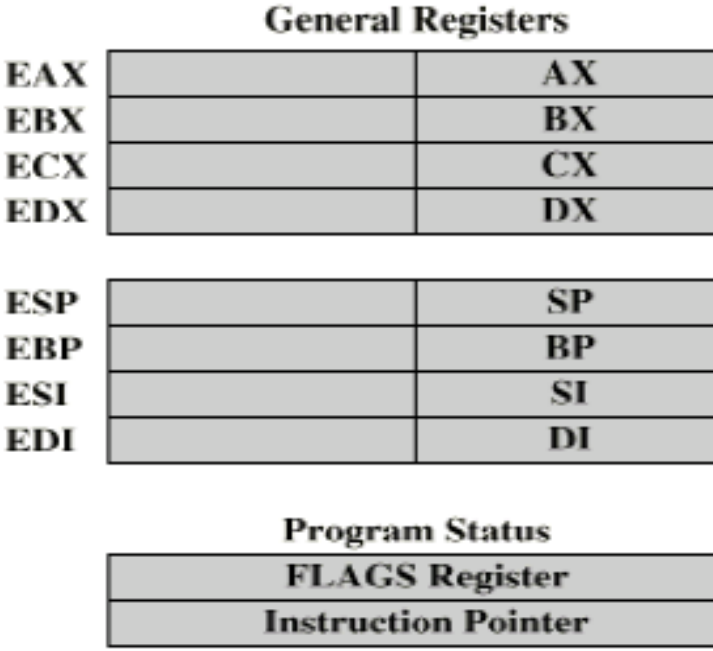
Example Register Organizations



(a) MC68000



(b) 8086



(c) 80386 - Pentium II

General Purpose Registers

- May be true general purpose
- May be restricted
- May be used for data or addressing
- Data
  - Accumulator
- Addressing
  - Segment

- Why make them general purpose?
  - Increase flexibility and programmer options
  - Increase instruction size & complexity

Why make them general purpose?

- –Increase flexibility and programmer options
- –Instruction size & complexity.

How big?

- Large enough to hold full address
- Large enough to hold full word
- Often possible to combine two data registers
  - C programming
  - double int a;
  - long int a;

Condition Code Registers(Flag Reg)

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

Control & Status Registers

- Program Counter (PC)
- Instruction Decoding Register(IR)
- Memory Address Register(MAR)
- Memory Buffer Register(MBR)

Registers

- **Memory Address Register (MAR)**
  - Connected to address bus
  - Specifies address for read or write op
- **Memory Buffer Register (MBR)**
  - Connected to data bus
  - Holds data to write or last data read
- **Program Counter (PC)**
  - Holds **address of next** instruction to be fetched
- **Instruction Register (IR)**
  - Holds last instruction fetched/current instruction being executed

Program Status Word

- A set of bits
- Includes Condition Codes
- Sign of last result
- Zero
- Carry
- Equal
- Overflow
- Interrupt enable/disable
- Supervisor

General Registers

- **AX is the primary accumulator**; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.
- **BX is known as the base register**, as it could be used in indexed addressing.
- **CX is known as the count register**, as the ECX, CX registers store the loop count in iterative operations.
- **DX is known as the data register**. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values.

Pointer Registers

- **Instruction Pointer (IP)** − The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.
- **Stack Pointer (SP)** − The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.
- **Base Pointer (BP)** − The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

Index Registers

SI and DI, are used for indexed addressing and sometimes used in addition and subtraction.

There are two sets of index pointers −

Source Index (SI) − It is used as source index for string operations.

Destination Index (DI) − It is used as destination index for string operations.

Control Registers

- The 32-bit instruction pointer register and the 32-bit flags register combined are considered as the control registers.
- Many instructions involve comparisons and mathematical calculations and change the status of the flags and some other conditional instructions test the value of these status flags to take the control flow to other location.
- The common flag bits are:

FLAG REGISTER 8086

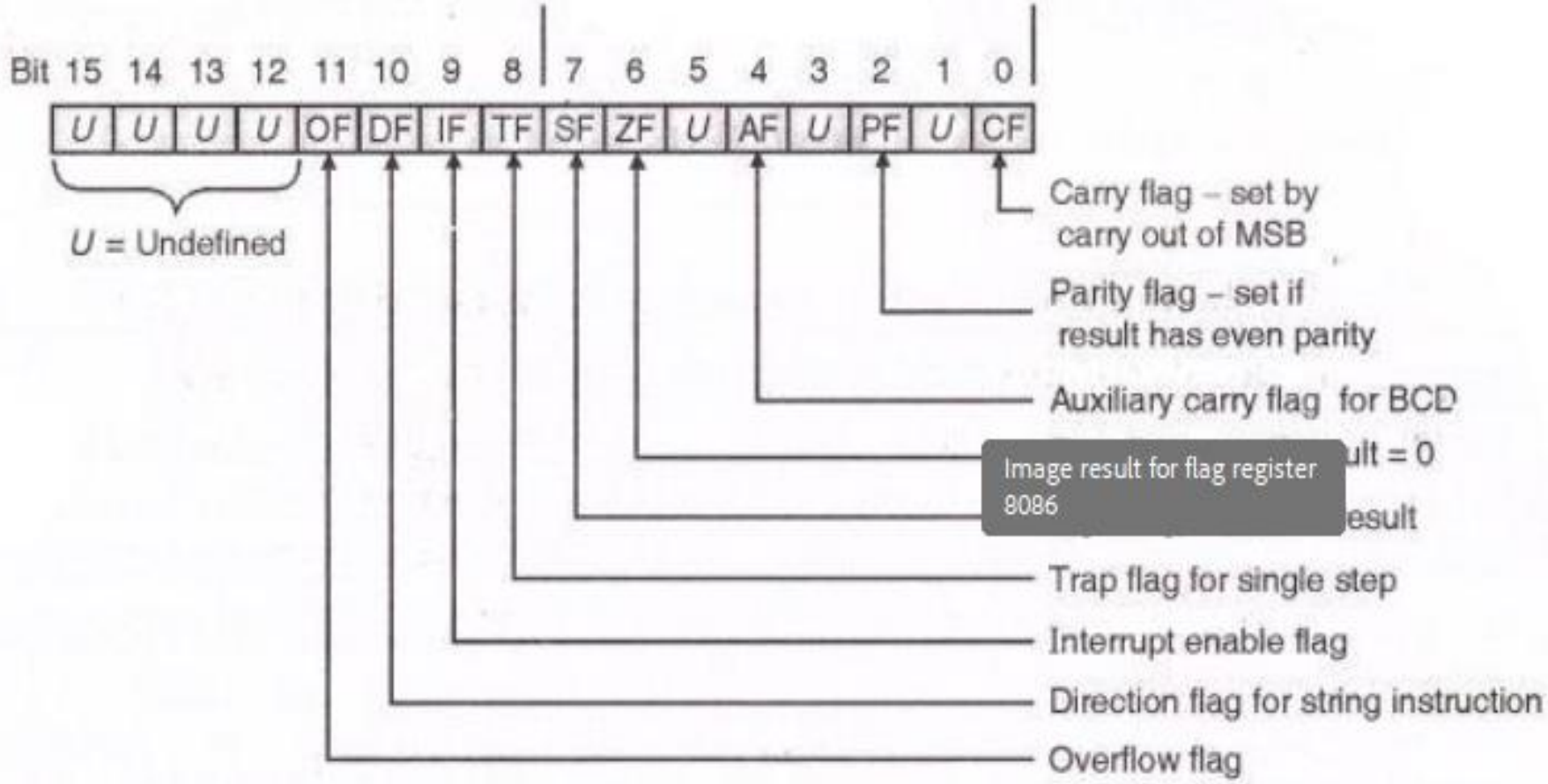


Image result for flag register 8086

8086 flag register format

- **Overflow Flag (OF) –** It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.
- **Direction Flag (DF) –** It determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.

- **Trap Flag (TF)** – It allows setting the operation of the processor in single-step mode. The DEBUG program we used sets the trap flag, so we could step through the execution one instruction at a time.
- **Sign Flag (SF)** – It shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item following the arithmetic operation. The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.
- **Zero Flag (ZF)** – It indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.

- **Interrupt Flag (IF)** – It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.

- **Auxiliary Carry Flag (AF)** – It contains the carry from bit 3 to bit 4 following an arithmetic operation; used for specialized arithmetic. The AF is set when a 1-byte arithmetic operation causes a carry from bit 3 into bit 4.
- **Parity Flag (PF)** – It indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits sets the parity flag to 1.
- **Carry Flag (CF)** – It contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a *shift* or *rotate* operation.

Segment Registers

- Segments are specific areas defined in a program for containing data, code and stack. There are three main segments –
- **Code Segment** – It contains all the instructions to be executed. A 16-bit Code Segment register or CS register stores the starting address of the code segment.
- **Data Segment(DS,ES)** – It contains data, constants and work areas. A 16-bit Data Segment register or DS register stores the starting address of the data segment.
- **Stack Segment** – It contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure. The Stack Segment register or SS register stores the starting address of the stack.

Instruction Formats

- Layout of bits in an instruction
- Includes **opcode**
- Includes (implicit or explicit) **operand(s)**
- Usually more than one instruction format in an instruction set

Instruction Length

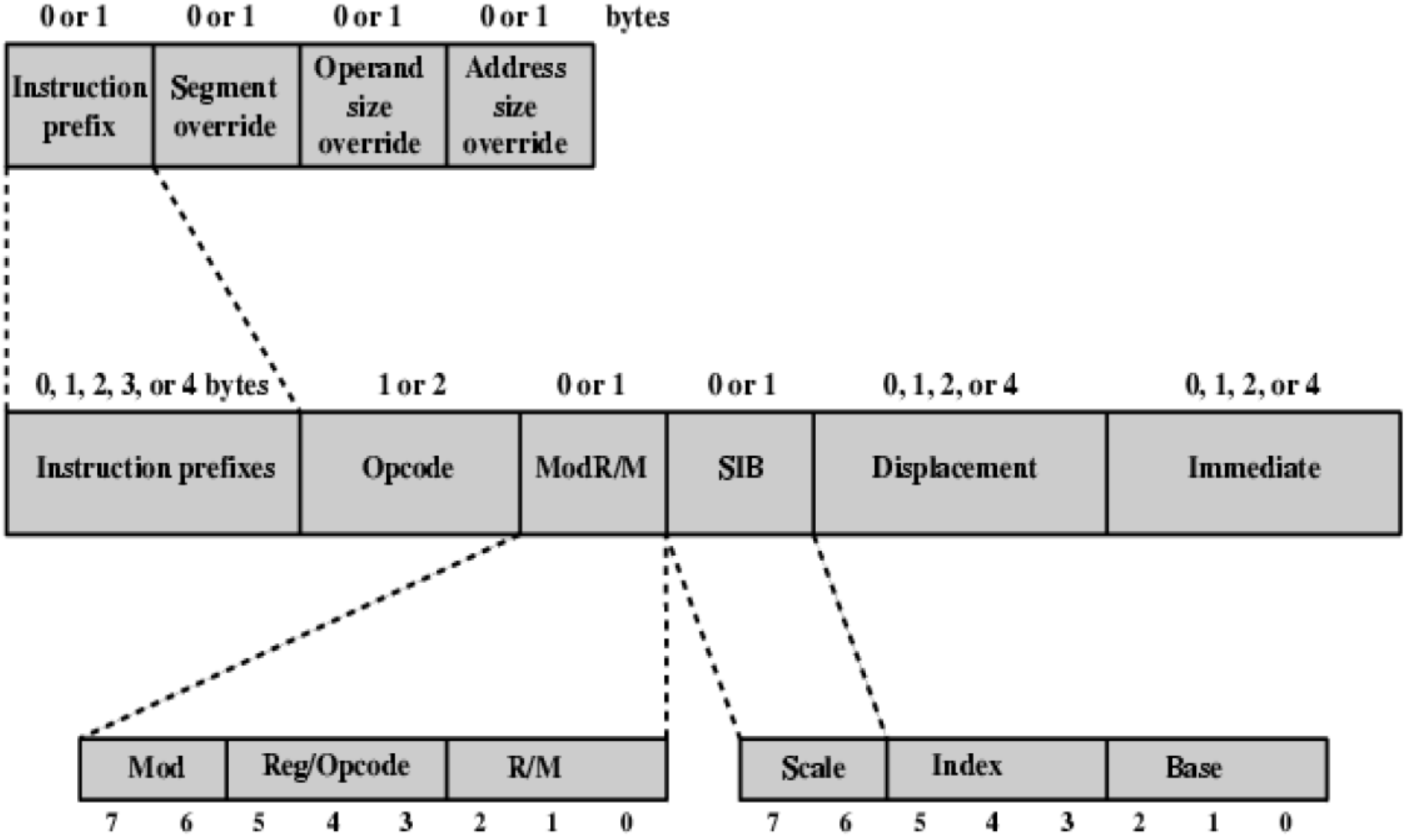
- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed

User wants more opcodes, operands, addressing modes , address range

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address Granularity

Pentium Instruction Format



PENTIUM INSTRUCTION FORMAT

- **Instruction Prefixes-LOCK** prefix or one of the **REPEAT** prefixes(REPE,REPNE,REPZ,REPNZ....)
- **Segment Override**-explicitly specifying segment register
- **Address Size**-16 or 32 bit (switch)
- **Operand Size**-16 or 32 bit (switch)

PENTIUM INSTRUCTION FORMAT

- **Opcode**
- **Mod R/m**-addressing
  - Mod+ r/m (combined info of registers and addressing modes)
  - Register-register or opcode info
- **SIB**
  - **Scale**-scale factor for scaled indexing
  - **Index**-specifies index register(SI,DI)
  - **Base**-specifies base register(BX)
- **Displacement**-8 or 16 or 32 bit
- **Immediate**-provides the value of 8/16/32 bit operand

William Stallings Computer Organization and Architecture 6th Edition

Chapter 11

Instruction Sets: Addressing Modes and Formats

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

Notations and Meanings

- In this section, we use the following notation:
- (X) = contents of memory location X or register X
- EA = actual (effective) address of the location containing the referenced operand
- R = contents of an address field in the instruction that refers to a register
- A = contents of an address field in the instruction

Immediate Addressing

- The simplest form of addressing is immediate addressing, in which the operand value is present in the instruction.

- This mode can be used to define and use constants or set initial values of variables.

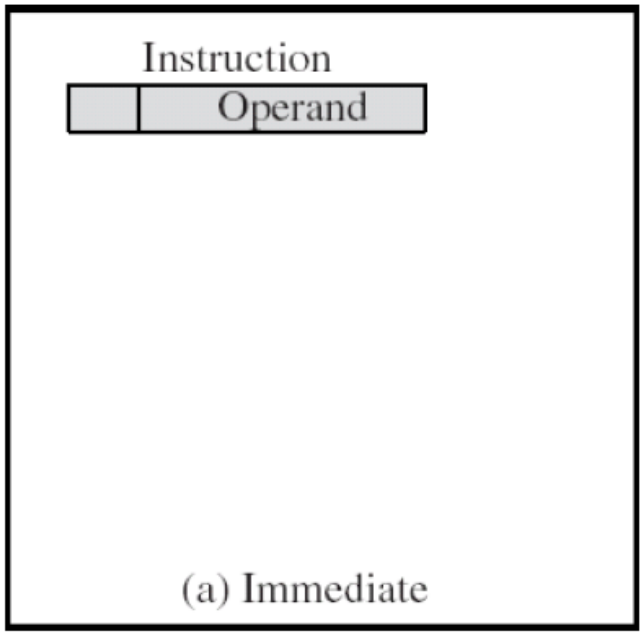
Immediate Addressing

- The **advantage** of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle.

- The **disadvantage** is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length.

Immediate Addressing

- In this mode data is present in address field of instruction .
- Designed like one address instruction format
  - Note:Limitation in the immediate mode is that the range of constants are restricted by size of address field.
- **FEATURES**
  - **Operand is part of instruction**
  - **Operand = address field**
  - **No memory reference to fetch data**
  - **Fast**
  - **Limited range**



MOV AX, 2000

MOV CL, 0A

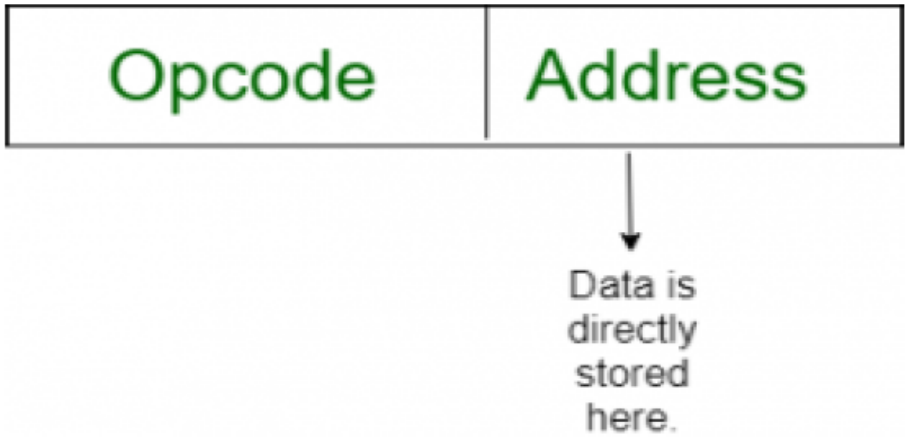
ADD AL, 45

ADD AX, 0000

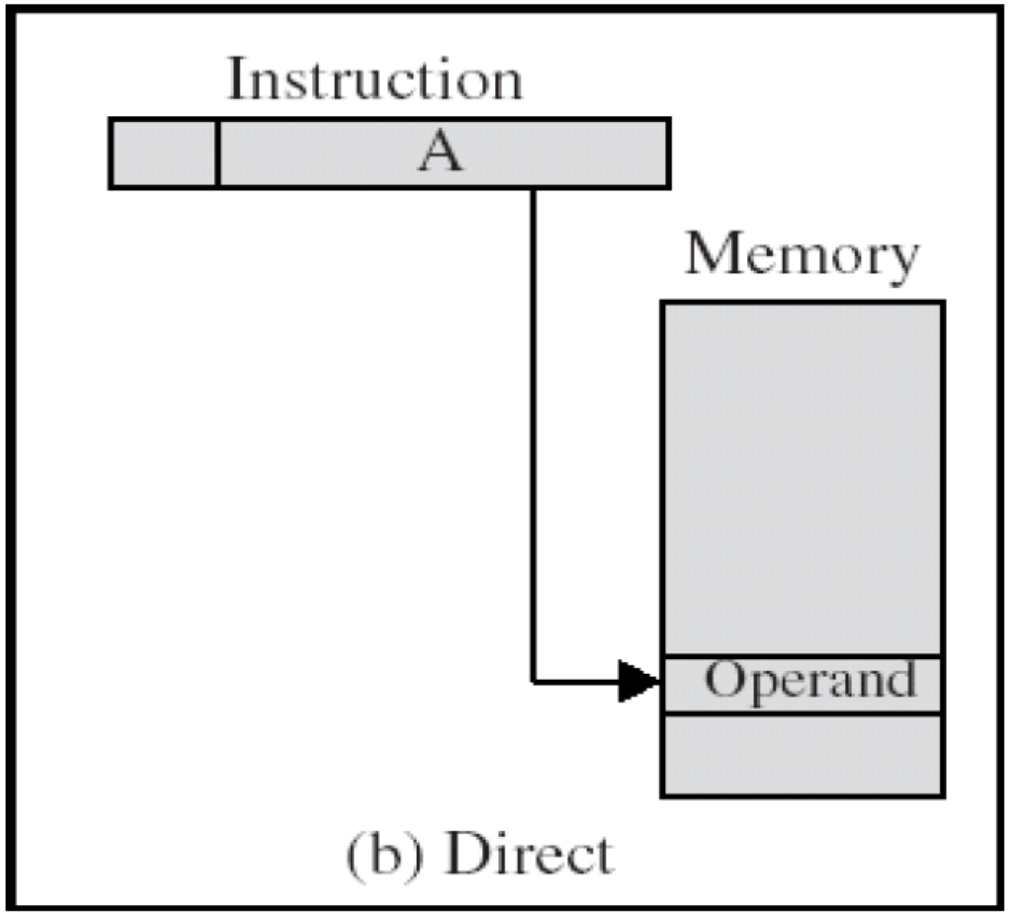
MOV CX, 4929 H

ADD AX, 2387 H,

MOV AL, FFH



Direct(M) Addressing Diagram



MOV AX, [DISP]

MOV AX, [0500]

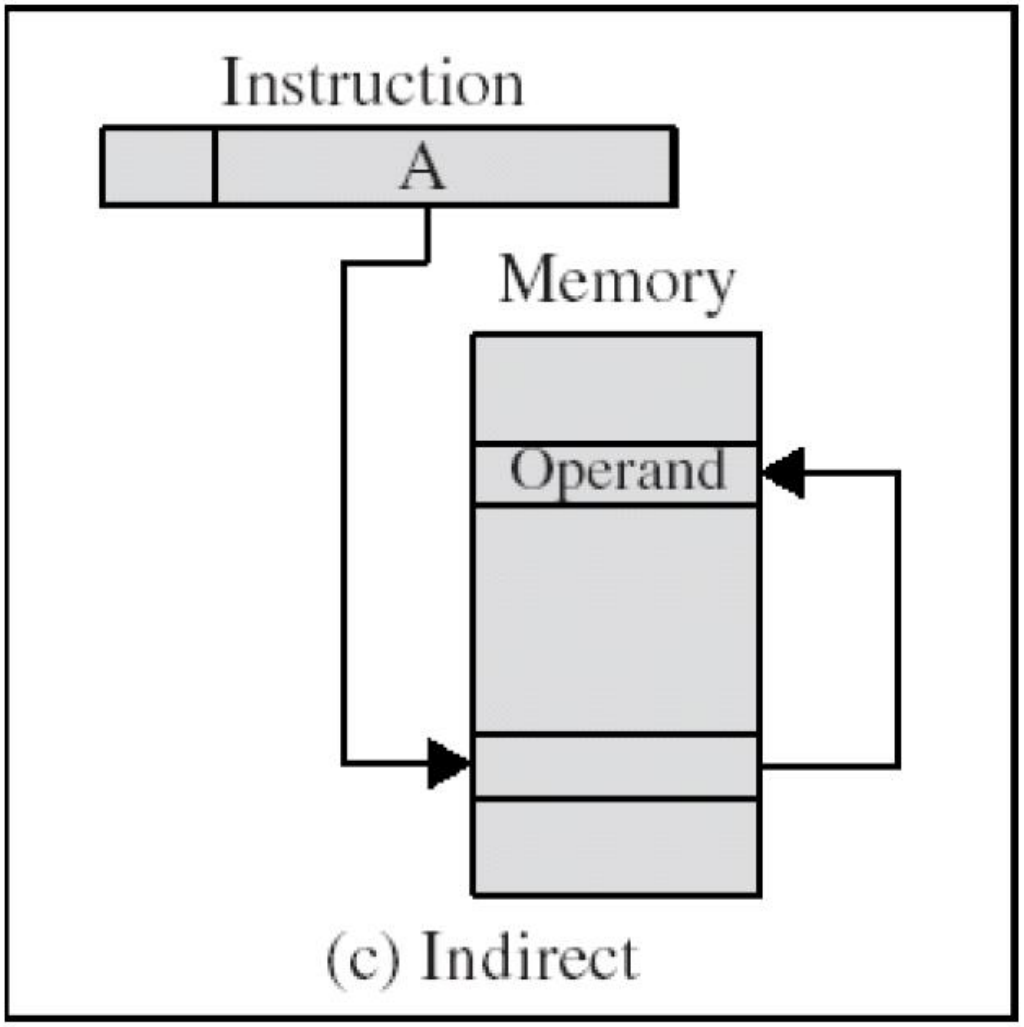
ADD AL,[0301]

- The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element.
- In this addressing mode the 16 bit effective address of the data is the part of the instruction.
- *Here only one memory reference operation is required to access the data.*

Direct Addressing

- Address field contains address of operand
- EFFECTIVE ADDRESS EA = address field (A)
  - Look in memory at address value for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Indirect Addressing Diagram



- In this mode address field of instruction contains the address of effective address.
- Here two references are required. 1st reference to get effective address. 2nd reference to access the data.

.

Based on the availability of Effective address, Indirect mode is of two kind:

- REGISTER INDIRECT: In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.
  - *Here one register reference,one memory reference is required to access the data.*

- MEMORY INDIRECT: In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.
  - *Here two memory reference is required to access the data*

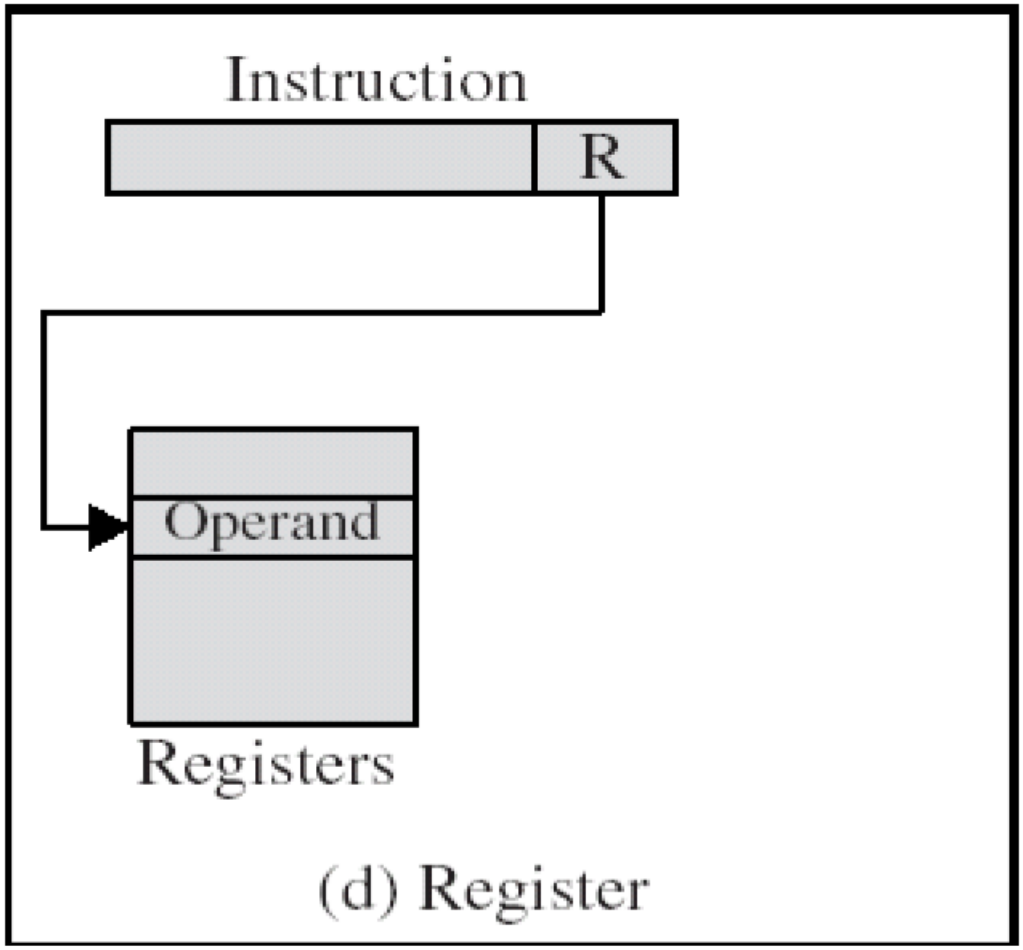
Indirect Addressing (1/2)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA =(A)
  - Look in A, find address (A) and look there for operand
- e.g. ADD AX, (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2/2)

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Register Addressing Diagram



- In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers.
- The data is in the register that is specified by the instruction.
  - *Here one register reference is required to access the data.*

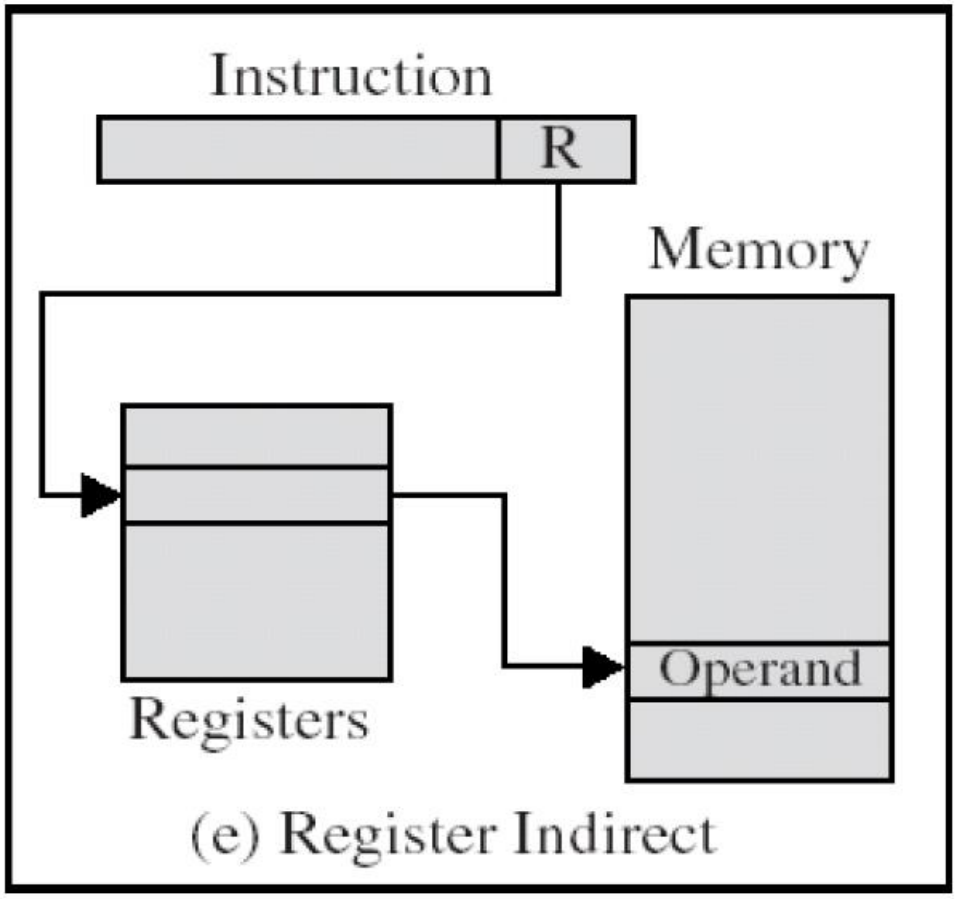
Register Addressing (1/2)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
  - MOV AX, BX
  - ADD AX, BX

Register Addressing (2/2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - N.B. C programming
    - register int a;
- c.f. Direct addressing

Register Indirect Addressing Diagram



- In this addressing the operand's offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction.
- The effective address of the data is in the base register or an index register that is specified by the instruction.
  - *Here two register reference is required to access the data.*

Register Indirect Addressing

MOV AX, [BX]

(move the contents of memory location s addressed by the register BX to the register AX)

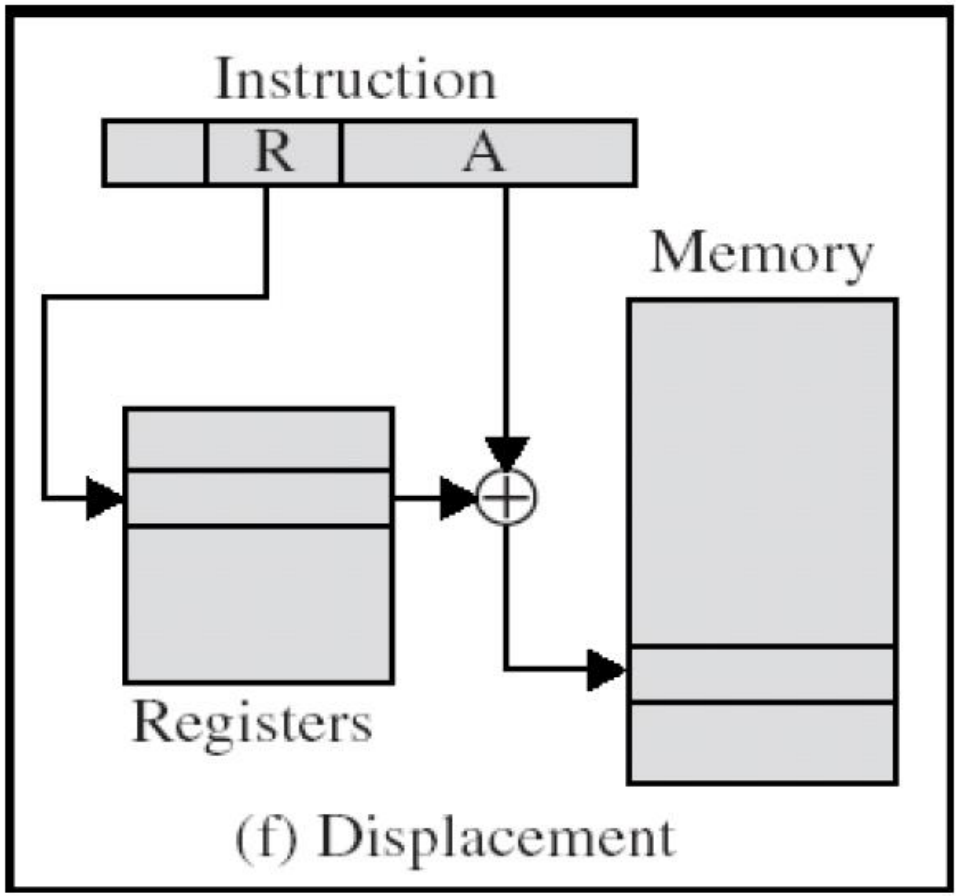
MOV AX, [DI]

ADD AL, [BX]

MOV AX, [SI]

- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

Displacement Addressing Diagram



Displacement Addressing

- EA = A + (R)
- Effective address=start address + displacement
- Effective address=Offset + (Segment Register)
- Use direct and register indirect
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

Base-Register Addressing

- Base register addressing mode is used to implement inter segment transfer of control.
- In this mode effective address is obtained by adding base register value to address field value.
- EA= Base register + Address field value.
- PC= Base register + Relative value.

Indexed Addressing

The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.

- MOV AX, [SI +05]

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add and push