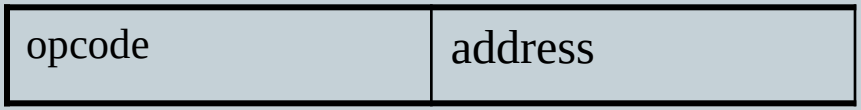
Instruction Set Architecture (ISA)

ADDRESSING MODES

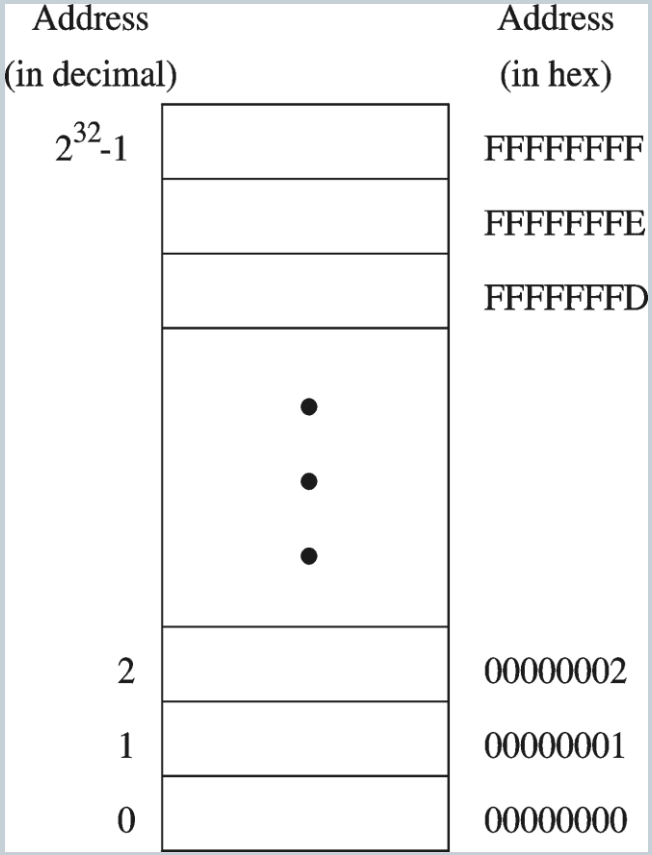
Addressing Mode



- Address field → address for operand & result
- Number of bit required to store data (operand & result)
  - E.g.: field size for an operand = 4 bit, 2^4=16 space for address can be used to store an operand
- How is the address of an operand specified?
  - Addressing mode –
    - A technique to identify address to access operands

Address Space: Main Memory (RAM)

- Memory is an ordered sequence of bytes
  - The sequence number is called the memory address
- Byte addressable memory
  - Each byte has a unique address
  - Supported by almost all processors
- Physical address space
  - Determined by the address bus width
  - Pentium has a 32-bit address bus
    - Physical address space = **4GB = 2^{32} bytes**
  - Itanium with a 64-bit address bus can support
    - Up to **16 Exabytes = 2^{64} bytes** of physical address space



Address Space is the set of memory locations (bytes) that can be addressed

Measures of capacity and speed:

- Kilo- (K) = 1 thousand = 10^3 and 2^{10}
- Mega- (M) = 1 million = 10^6 and 2^{20}
- Giga- (G) = 1 billion = 10^9 and 2^{30}
- Tera- (T) = 1 trillion = 10^{12} and 2^{40}
- Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}
- Exa- (E) = 1 quintillion = 10^{18} and 2^{60}
- Zetta- (Z) = 1 sextillion = 10^{21} and 2^{70}
- Yotta- (Y) = 1 septillion = 10^{24} and 2^{80}

Whether a metric refers to a power of ten or a power of two typically depends upon what is being measured.

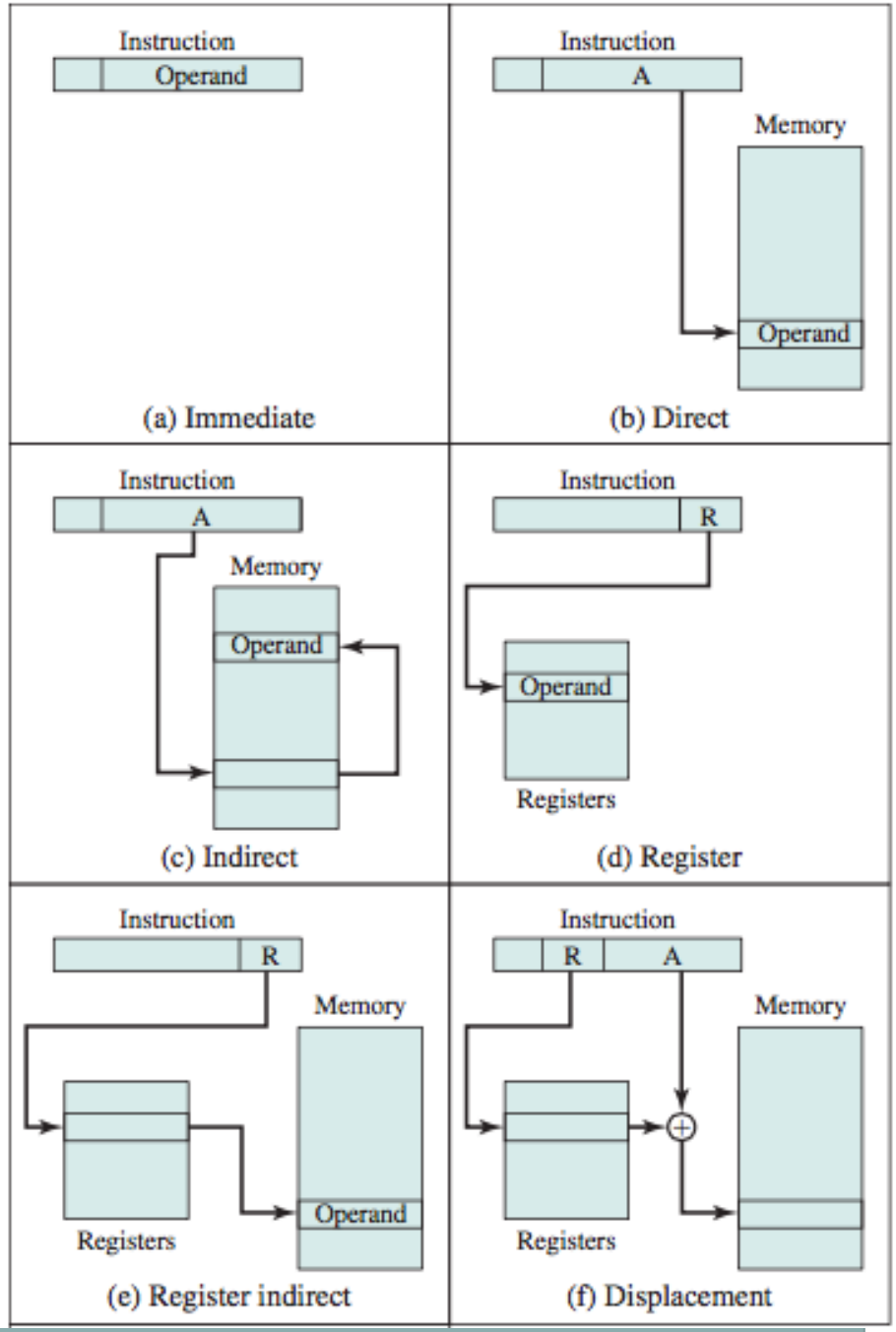
Addressing Modes

A=contents of an address field in instruction

R=contents of an address field in instruction that refers to a register

EA=actual (effective) address of the location containing the referenced operand

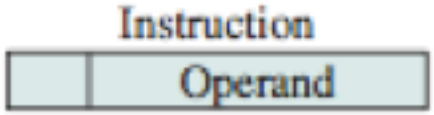
(X)=contents of memory location X or register X



Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Immediate Addressing



- Operand is part of instruction
- Operand = A
- Used to:
  - define and use constant
  - Set initial values of variables
- No memory reference to fetch data ➔ so it is FAST
- Size of the operand is limited to the size of address field

Opcode	Operand
ADD	AX , 5H

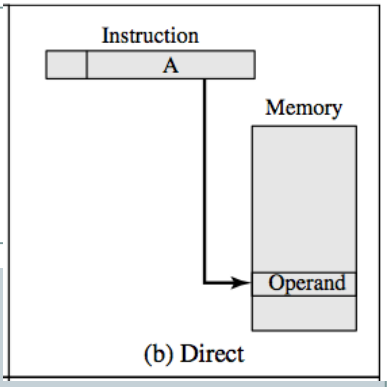
Immediate value

A = contents of an address field in instruction

Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Direct Addressing



- Address field contains *address of operand*
- Effective Address (EA) = address field of (A)
  - EA will be either a virtual memory (if present), main memory address or a register
- e.g. add EAX, A
  - Look in memory at address A for operand
  - Add contents of cell A to register EAX
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

```
add   AX , count
add   AX , (1011)
```

Direct Addressing

```
EAX=7611ED10  EBX=7FFD2210  ECX=00000123  EDX=00401022
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=0040102D  EFL=00000246  CF=0  SF=0  ZF=1  OF=0
```

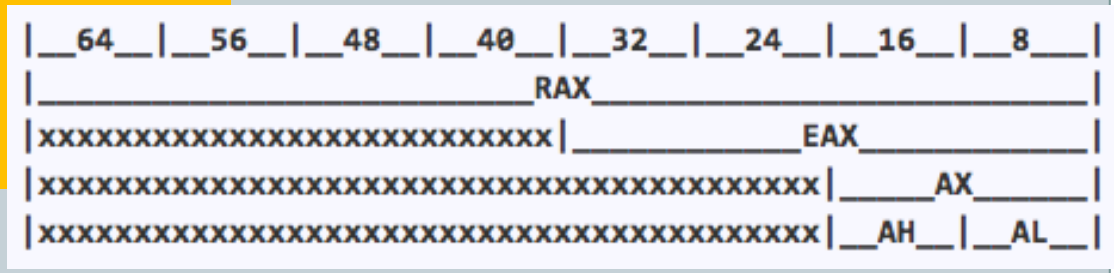
```
.data

val1 byte 10h
array1 word 2210h, 11h, 12h, 13h
array2 dword 123h, 234h, 345h, 456h


.code
main PROC

        mov AL, val1
        mov BX, array1
        mov ECX, array2
        call dumpregs

        exit
main ENDP
```

AL = 10h

BX = 2210h

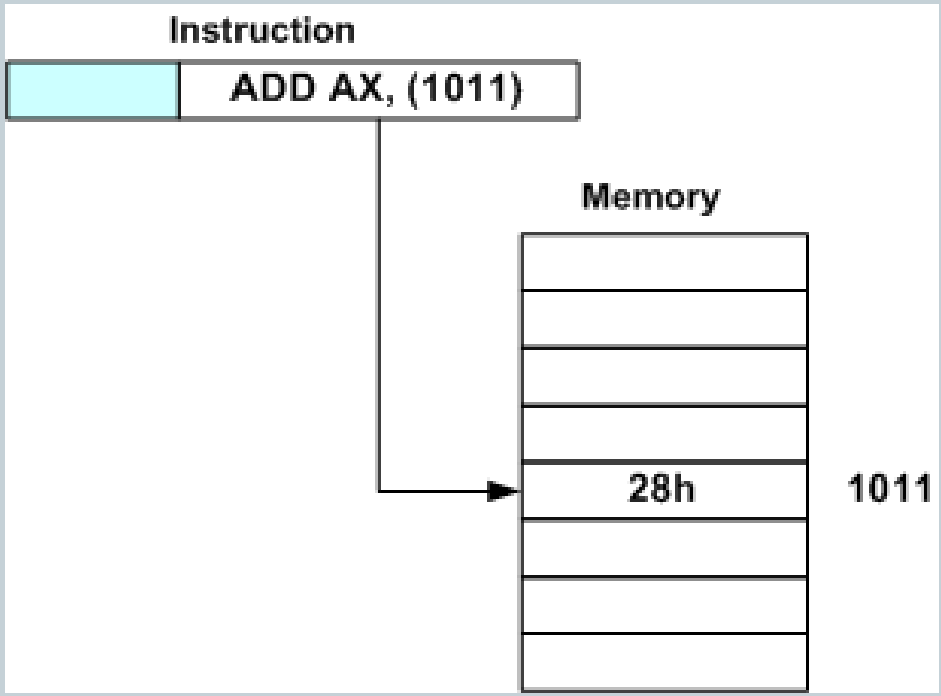
ECX = 00000123h

- Variables are defined in the data section of the program
- We use the variable name (label) to address memory directly
- Assembler computes the offset of a variable
- The variable offset is specified directly as part of the instruction



Direct Addressing

ADD	AX , (1011)



AX = 10H

AX = 10H + 28H = 38H

Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Register Addressing

- Similar to direct addressing
- BUT, the address field refers to a register
- EA = **R**

R = contents of an address field in instruction that refers to a register

- Limited number of registers ➔ compared to memory locations

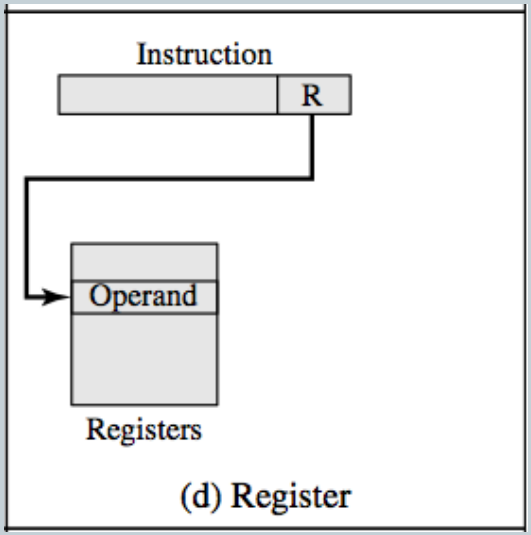
Register Addressing

```
.code
main PROC

        mov EAX, 0
        mov EBX, 2000h
        mov ECX, 3000h

        mov EAX, EBX
        add EAX, ECX

        call dumpregs

        exit
main ENDP
```

```
EAX=00005000  EBX=00002000  ECX=00003000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0012FF94  ESP=0012FF8C
EIP=00401028  EFL=00000206  CF=0  SF=0  ZF=0  OF=0
```

EAX = 00002000h

EAX = 00005000h



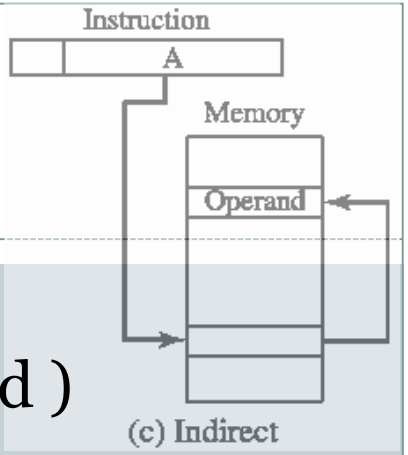
Register Addressing

- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No time consuming for memory references ➔ faster
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing

Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Indirect Addressing



- Memory cell pointed to by address field contains the address of the operand @ (pointer to operand)
- EA = [A]
  - Look in A, find address [A] and look there for operand
- e.g.

```
.data
byteVal BYTE 10h
.code
mov  esi,OFFSET byteVal
mov  al,[esi]                    ; AL = 10h
```

  - Add contents of cell pointed by ESI to register AL

Indirect Addressing

```
EAX=00000234   EBX=00000010   ECX=00000123   EDX=00123456
ESI=00000000   EDI=00000000   EBP=0018FF98   ESP=0018FF90
EIP=00401042   EFL=00000246   CF=0   SF=0   ZF=1   OF=0   AF=0   PF=1
```

- Indirect operands are ideal for traversing an array:

```
.data
array1 byte 10h, 11h, 12h, 13h
array2 word 123h, 234h, 345h, 456h
array3 dword 123456h, 23456789h

.code
main PROC
        mov BL, [array1]
        mov CX, [array2]
        mov EDX, [array3]
        mov AX, [array2 + 2]
        call dumpregs
        exit
main ENDP
```

Move the content of the memory where the first byte of array1 is kept into BL

BL = 10h

CX = 0123h

EDX = 00123456h

AX = 0234h

	Memory address	Memory content
array1	00404000	10
	00404001	11
	00404002	12
	00404003	13

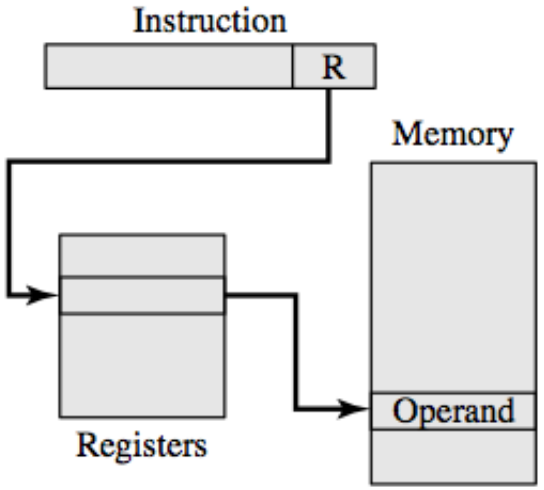
- **Note that the register in brackets must be incremented by a value that matches the array type ➔ 1 for byte, 2 - word, 4 - dword.

Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Register Indirect Addressing

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



(e) Register indirect

Code 1: Register Indirect Addressing

```
EAX=00000234   EBX=00000010   ECX=00000123   EDX=00404004
ESI=00404004   EDI=00404008   EBP=0012FF94   ESP=0012FF8C
EIP=0040103E   EFL=00000246   CF=0   SF=0   ZF=1   OF=0
```

```
.data
array1 byte 10h, 11h, 12h, 13h
array2 word 123h, 234h, 345h, 456h
array3 dword 123456h, 23456789h

.code
main PROC
        mov ESI, OFFSET array1
        mov EDI, OFFSET array2

        mov BL, [ESI]
        mov CX, [EDI]
        mov EDX, (ESI)
        mov AX, [EDI + 2]
        call dumpregs
        exit
main ENDP
```

The address that holds the first byte of array1 is stored into register ESI ➔ must be 32 bit register

Read the register ESI, go to memory address, get the value , store in BL.

ESI = 00404004
EDI = 00404008

	Memory address	Memory content
array1	00404004	10
	00404005	11
	00404006	12
	00404007	13

Code 1: Register Indirect Addressing

```
EAX=00000234   EBX=00000010   ECX=00000123   EDX=00404004
ESI=00404004   EDI=00404008   EBP=0012FF94   ESP=0012FF8C
EIP=0040103E   EFL=00000246   CF=0   SF=0   ZF=1   OF=0
```

```
.data
array1 byte 10h, 11h, 12h, 13h
array2 word 123h, 234h, 345h, 456h
array3 dword 123456h, 23456789h

.code
main PROC
        mov ESI, OFFSET array1
        mov EDI, OFFSET array2

        mov BL, [ESI]
        mov CX, [EDI]
        mov EDX, (ESI)
        mov AX, [EDI + 2]
        call dumpregs
        exit
main ENDP
```

Read the register EDI, go to memory address, get the value , store in CX.

ESI = 00404004
EDI = 00404008

	Memory address	Memory content
array2	00404008	23
	00404009	01
	00404010	34
	00404011	02

Code 1: Register Indirect Addressing

```
EAX=00000234   EBX=00000010   ECX=00000123   EDX=00404004
ESI=00404004   EDI=00404008   EBP=0012FF94   ESP=0012FF8C
EIP=0040103E   EFL=00000246   CF=0   SF=0   ZF=1   OF=0
```

```
.data
array1 byte 10h, 11h, 12h, 13h
array2 word 123h, 234h, 345h, 456h
array3 dword 123456h, 23456789h

.code
main PROC
        mov ESI, OFFSET array1
        mov EDI, OFFSET array2

        mov BL, [ESI]
        mov CX, [EDI]
        mov EDX, (ESI)
        mov AX, [EDI + 2]
        call dumpregs
        exit
main ENDP
```

Read the register EDI, add a word (OR add 2 bytes), go to memory address shown, get the value of second element of array2 , store in AX.

ESI = 00404004
EDI = 00404008

	Memory address	Memory content
array2	00404008	23
	00404009	01
	00404010	34
	00404011	02

Addressing Modes

- The most basic addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)

Displacement Addressing

- Combines direct addressing and register indirect addressing
- EA = A + [R]
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa
- 3 common displacement addressing technique:
  - Indexed addressing
  - Relative addressing
  - Base register addressing

Indexed Addressing

- A = base
- R = displacement
- EA = A + R
- Good for *accessing @ traversing arrays*

Indexed Addressing: Array Traversal

```
.data
array1 byte 10h, 11h, 12h, 13h
array2 word 123h, 234h, 345h, 456h
array3 dword 123456h, 23456789h

.code
main PROC
        mov EAX, 0
        mov ECX, 4

        L1:
        mov BX, array2[EAX]
        add EAX,2
        call DumpRegs
        loop L1

        exit
main ENDP
```

```
EAX=00000002  EBX=00000123  ECX=00000004  EDX=00000000
ESI=00404004  EDI=00404008  EBP=0012FF94  ESP=0012FF8C
EIP=00401047  EFL=00000202  CF=0  SF=0  ZF=0  OF=0

EAX=00000004  EBX=00000234  ECX=00000003  EDX=00000000
ESI=00404004  EDI=00404008  EBP=0012FF94  ESP=0012FF8C
EIP=00401047  EFL=00000202  CF=0  SF=0  ZF=0  OF=0

EAX=00000006  EBX=00000345  ECX=00000002  EDX=00000000
ESI=00404004  EDI=00404008  EBP=0012FF94  ESP=0012FF8C
EIP=00401047  EFL=00000206  CF=0  SF=0  ZF=0  OF=0

EAX=00000008  EBX=00000456  ECX=00000001  EDX=00000000
ESI=00404004  EDI=00404008  EBP=0012FF94  ESP=0012FF8C
EIP=00401047  EFL=00000202  CF=0  SF=0  ZF=0  OF=0
```

base

displacement

similar to:

```
mov BX, [array2 + EAX]
add EAX, 2
```

Activity : Array Traversal

- First, open new 'Template' project in VS2010 and copy & paste source from elearning, i.e. *Relative Addressing (Array Traversal)*
  - *Refer to steps described in Lab 1 session*
- Then proceed with these 2 tasks:
  - (A) Execute ; [Start Without Debugging]
  - (B) Debug ; F10

Activity : Array Traversal

- 2 tasks:
  - (A) Execute ; [Start Without Debugging]
    - Looking at changes of registers value, i.e.;
      - EAX
      - EBX
      - ECX

Activity : Array Traversal

- 2 tasks:
  - (B) Debug ; press **F10**
    - Open 3 debug windows, i.e.;
      - Registers
      - Memory 1
      - Disassembly
        - For { Disassembly } window, enable all options in [Viewing Options]

… end of Part 4

- Hierarchy of Computer Languages
- General Concepts
- ISA Level
- Elements of Instructions
- Instructions Types
- Instruction Formats
- Number of Addresses
- Registers
- Types of Operands
- **Addressing Modes**

Part 4

Summary

MODULE 4:
INSTRUCTION SET ARCHITECTURE

Instruction Set Design Decisions

Very complex because it affects so many aspects of the computer system

Defines many of the functions performed by the processor

Programmer's means of controlling the processor

Fundamental design issues:

Operation repertoire	Data types	Instruction format	Registers	Addressing

Instruction Set Design Decisions

Fundamental design issues:

Operation repertoire

- How many and which operations to provide and
- how complex operations should be

Instruction Types: Categories

- Data processing
  - Arithmetic and logic instructions
- Data storage (main memory)
  - Memory instructions
    - movement of data into or out of memory locations
- Data movement (I/O)
  - I/O instructions
- Control (Program flow control)
  - Test and branch instructions

Instruction Set Design Decisions

Fundamental design issues:

Data types

The various types of data upon which operations are performed

- Most important general categories of data are:
  - Numbers – numeric data
    - Integer/floating point/decimal
    - Limited magnitude of numbers – integer/decimal
    - Limit precision – floating point
  - Characters – data for text and strings
    - ASCII, UNICODE etc.
  - Logical Data
    - Bits or flags

Instruction Set Design Decisions

Fundamental design issues:

Instruction format

Instruction length in bits, number of addresses, size of various fields, etc.

x86 Instruction Format

Opcode	ModR/M	SIB	displacement	Immediate
1 or 2 bytes	1 byte, If required	1 byte, If required	1,2 or 4 bytes If required	1,2 or 4 bytes If required

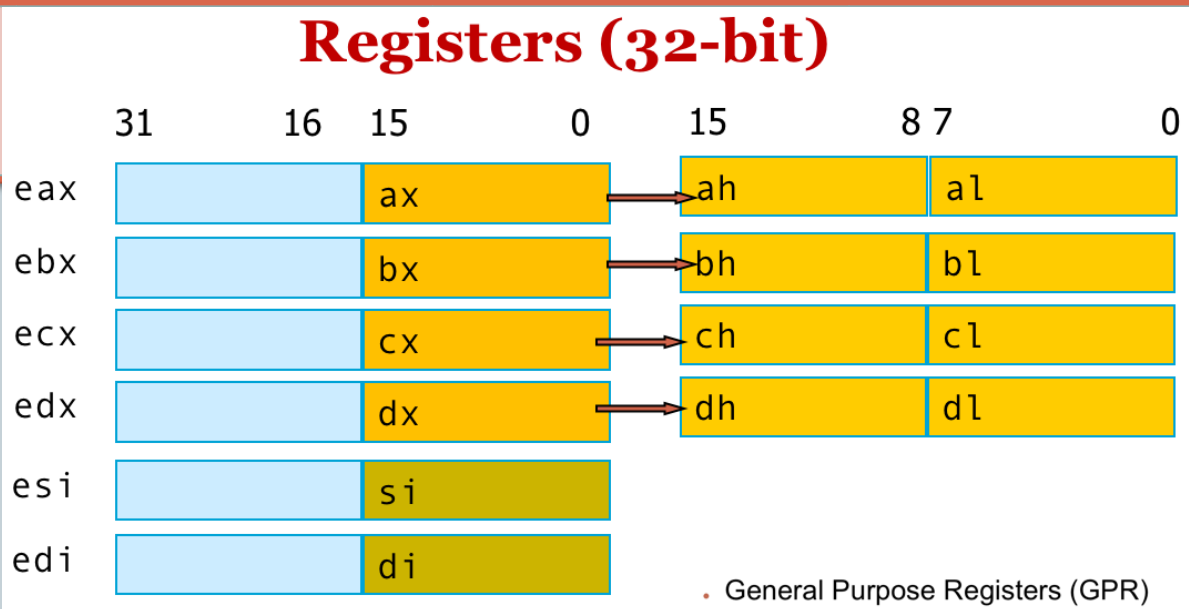
- Opcode: determine the action
- ModR/M: Addressing modes register/memory
- SIB: Scale-Index-Base

Instruction Set Design Decisions

Fundamental design issues:

Registers

Number of processor registers that can be referenced by instructions and their use



- General Purpose Registers (GPR)
  - AX, BX, CX, DX
- Pseudo General Purpose Registers
  - Stack: SP (stack pointer), BP (base pointer)
  - Strings: SI (source index), DI (destination index)
- Special Purpose Registers
  - IP (instruction pointer) and EFLAGS

Instruction Set Design Decisions

Fundamental design issues:

Addressing

The mode or modes by which the address of an operand is specified

Addressing Modes

A=contents of an address field in instruction

R=contents of an address field in instruction that refers to a register

EA=actual (effective) address of the location containing the referenced operand

(X)=contents of memory location X or register X