Computer Organization and Software Systems

Session 5

Dr. Lucy J. Gudino
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

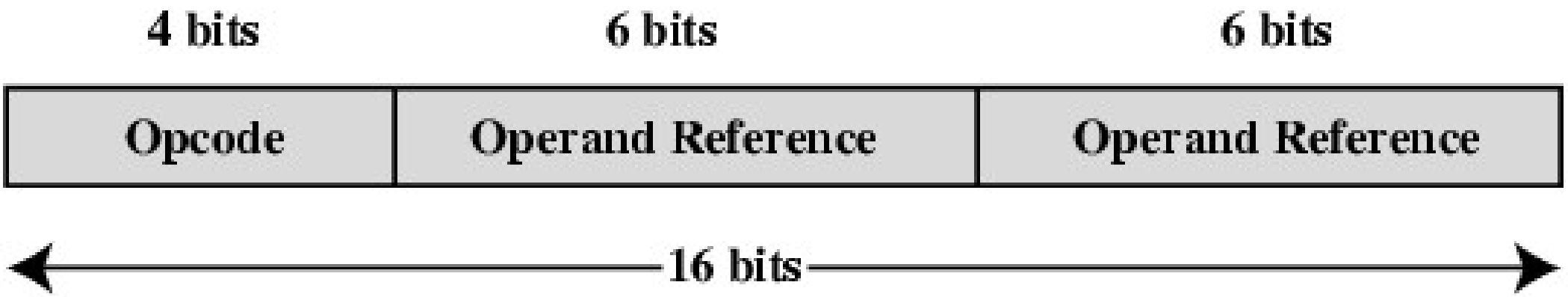
CISC Instruction Set (Intel x86 as an example)

BITS Pilani
Pilani Campus

Introduction

- What is an Instruction Set?
  - The complete collection of instructions that are understood by a CPU
- Elements of an Instruction
  - Operation code (Op code)
  - Source Operand reference
  - Result Operand reference
  - Next Instruction Reference
- Source and Destination Operands can be found in four areas
  - Main memory (or virtual memory or cache)
  - CPU register
  - Immediate
  - I/O device

Simple Instruction Format



- During instruction execution, an instruction is read into an instruction register (IR) in the processor.
- The processor must be able to extract the data from the various instruction fields to perform the required operation.
- Opcodes are represented by abbreviations, called ***mnemonics***

  Example: ADD AX, BX ➔ Add instruction

Instruction Types

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

Number of Addresses (1/2)

- 3 addresses
  - Operand 1, Operand 2, Result
  - c = a + b; add a, b, c
  - May be a forth - next instruction (usually implicit)
  - Needs very long words to hold everything
- 2 addresses
  - One address doubles as operand and result
  - a = a + b : add a, b
  - Reduces length of instruction
  - The original value of a is lost.

Number of Addresses (2/2)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. c = a + b

```
push a
push b
add
pop c
```

Example

Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

How Many Addresses

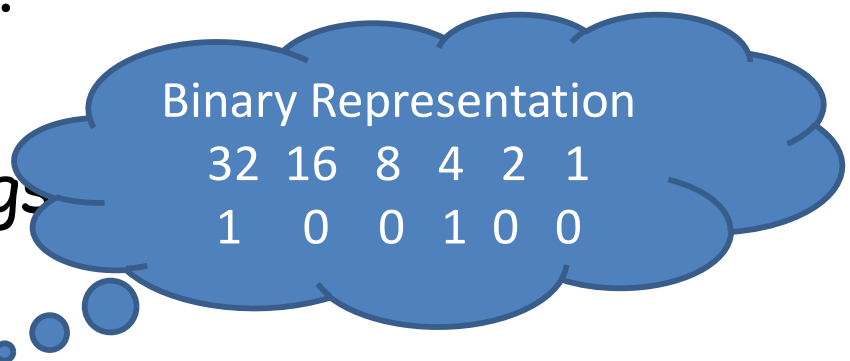
- Fewer addresses
  - More Primitive instructions, shorter length instructions
  - Less complex instructions, hence requires less complex hardware
  - More instructions per program
    - Longer programs
    - More complex programs
    - Longer execution time
- Multiple address instructions
  - Lengthy instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program

Instruction set Design Decisions

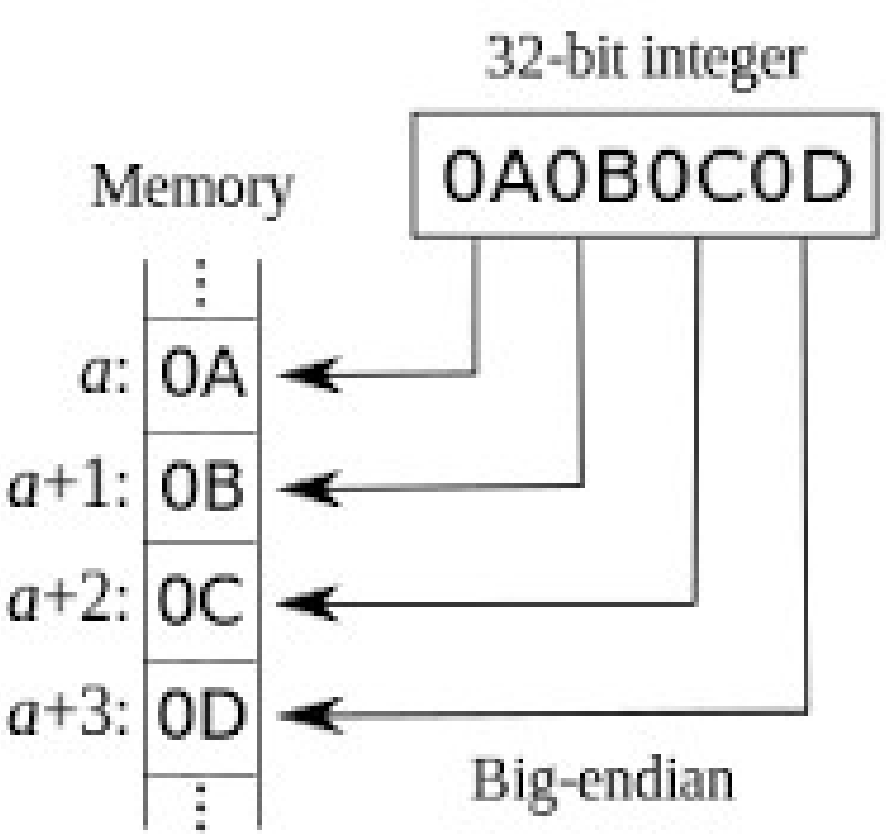
- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes

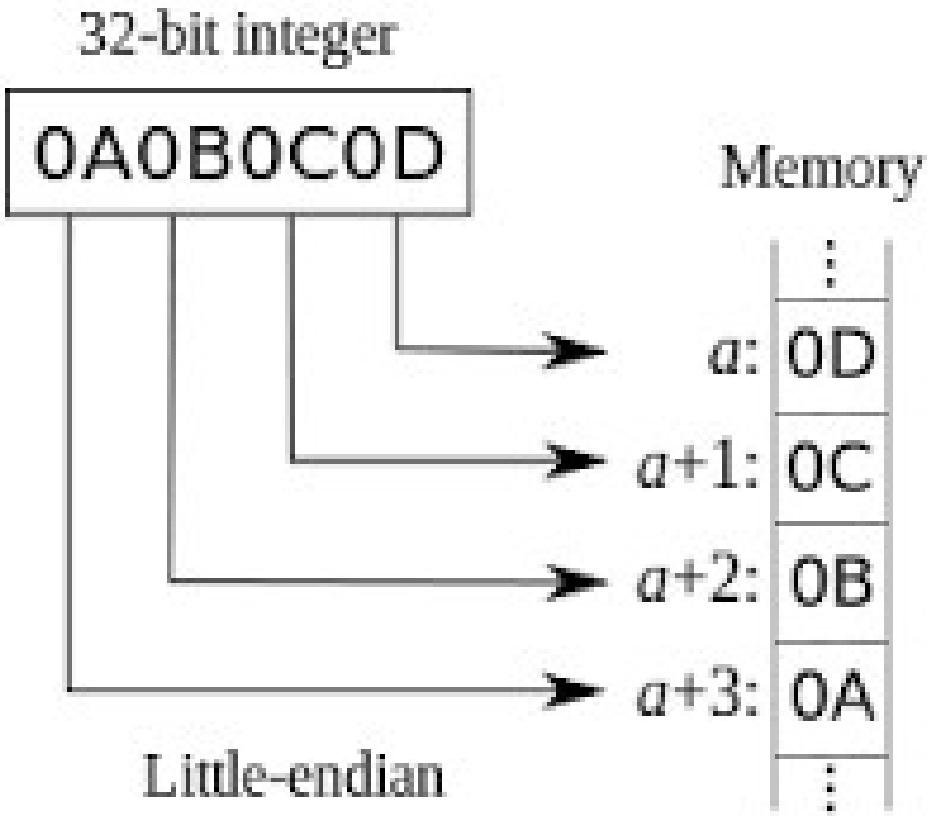
Types of Operand

- Machine instructions operate on data
- General categories of data
  - Addresses
  - Numbers
    - Binary integer or binary fixed point, floating point, decimal
  - Characters
    - ASCII etc.
  - Logical Data
    - Bits or flags
- Packed Decimal
  - 36 : 0011 0110



Byte Ordering

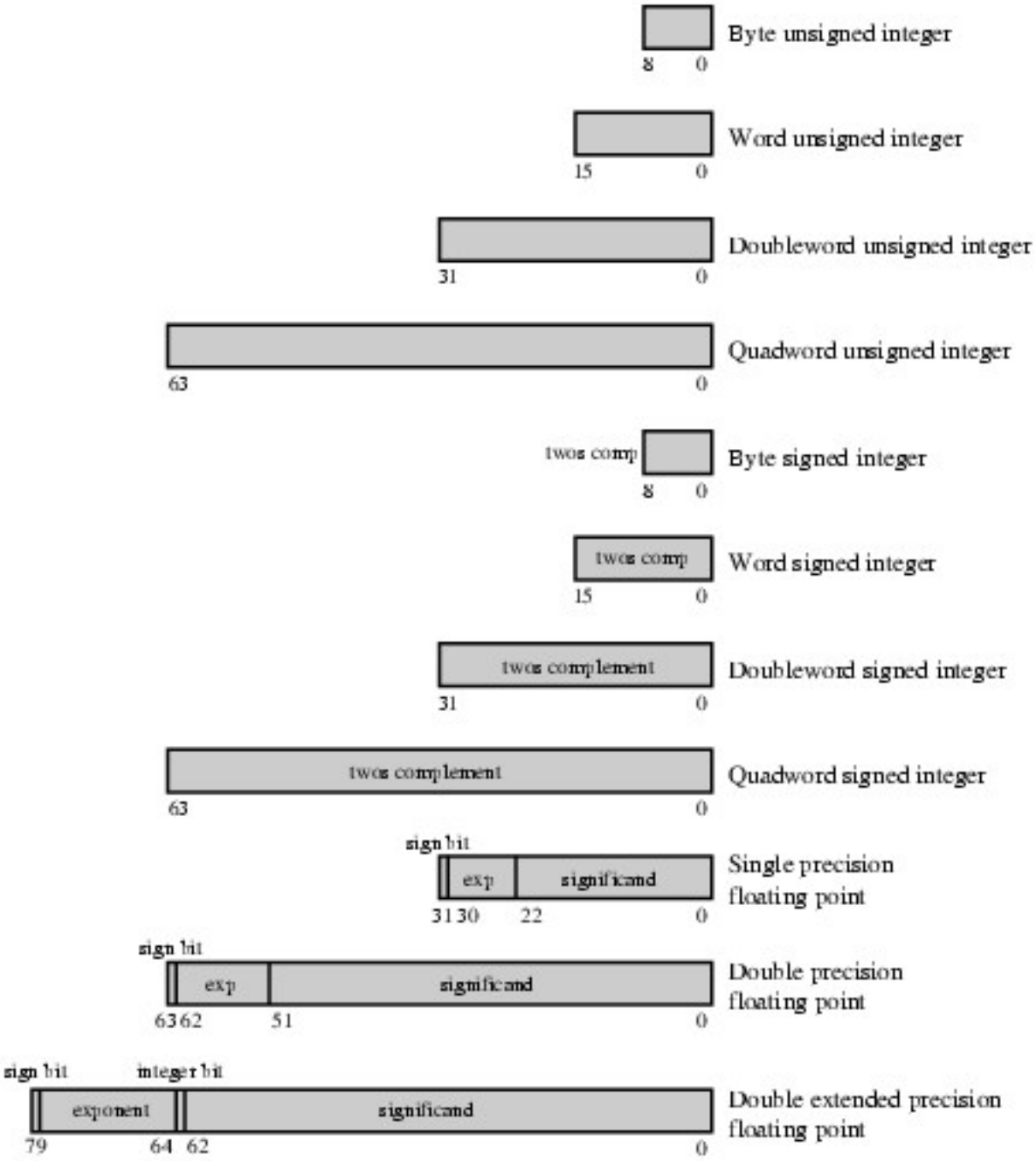




x86 Data Types

- General – Byte, Word, double word, quadword, double quad word - arbitrary binary contents
- Integer – signed binary using two's complement representation
- Ordinal - unsigned integer
- Unpacked BCD - One digit per byte
- Packed BCD - 2 digits per byte
- Near Pointer
- Far pointer
- Bit field : A contiguous sequence of bits in which the position of each bit is considered as an independent unit.
- Bit and Byte String
- Floating Point

x86 Numeric Data Formats



Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Data Transfer

- Specify
  - Source
  - Destination
  - Amount of data
- Action:

1. Calculate the memory address, based on the address mode
2. If the address refers to virtual memory, translate from virtual to real memory address.
3. Determine whether the addressed item is in cache.
4. If not, issue a command to the memory module.

Arithmetic

- Add, Subtract, Multiply, Divide
- May include
  - Absolute value (|a|)
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)
- Signed Integer
- Floating point

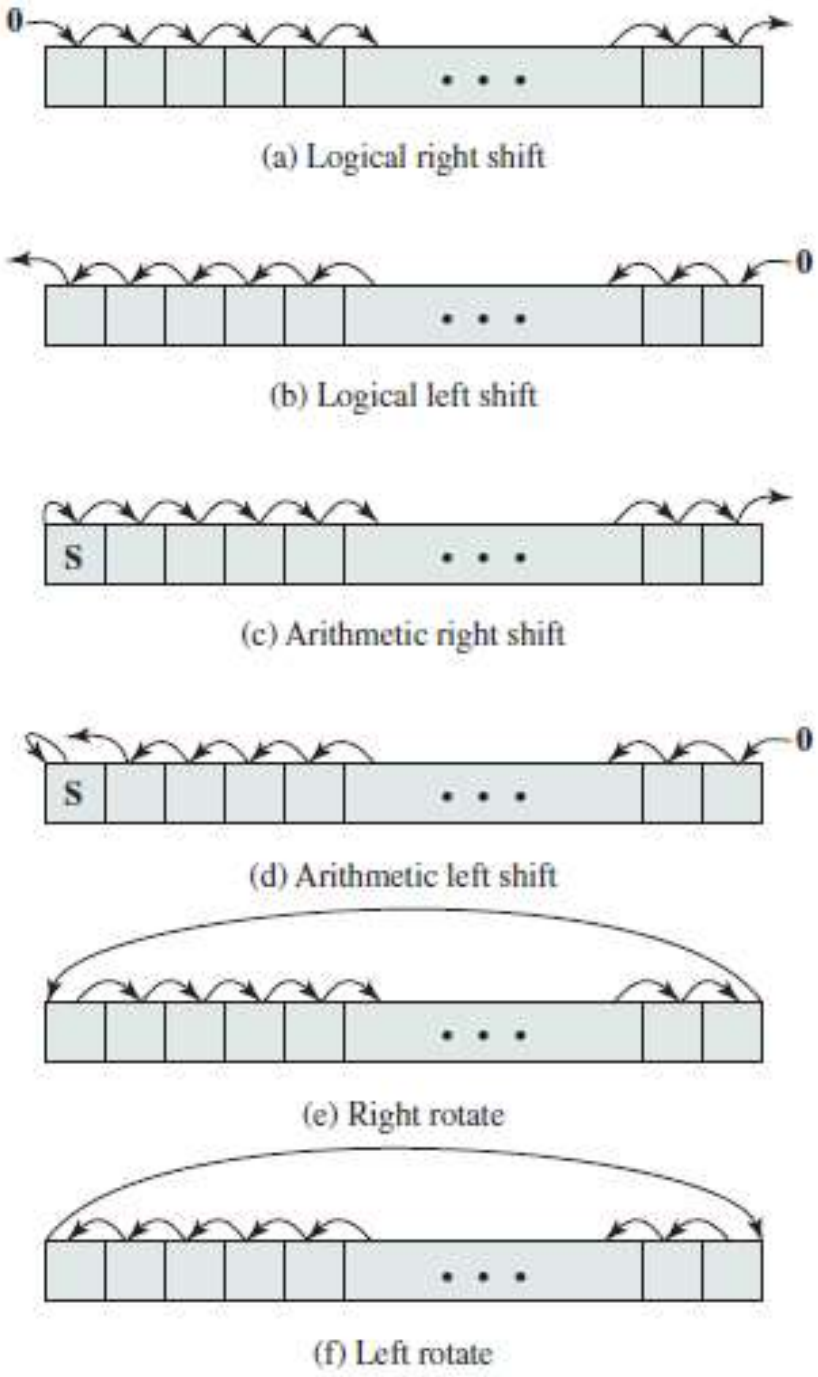
Logical

- Bitwise operations
- AND, OR, XOR, NOT

Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Shift and Rotate Operations



Input	Operation	Output
10101101	Logical right shift (3 bits)	
10101101	Logical left shift (3 bits)	
10101101	Arithmetic right shift (3 bits)	
10101101	Arithmetic left shift (3 bits)	
10101101	Right rotate (3 bits)	
10101101	Left rotate (3 bits)	

Conversion

E.g. Binary to Decimal

Input/Output

- May be specific instructions (I/O-Mapped I/O)
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

Systems Control

- Privileged instructions
- CPU needs to be in specific state
  - User Mode
  - Kernel mode
- For operating systems use

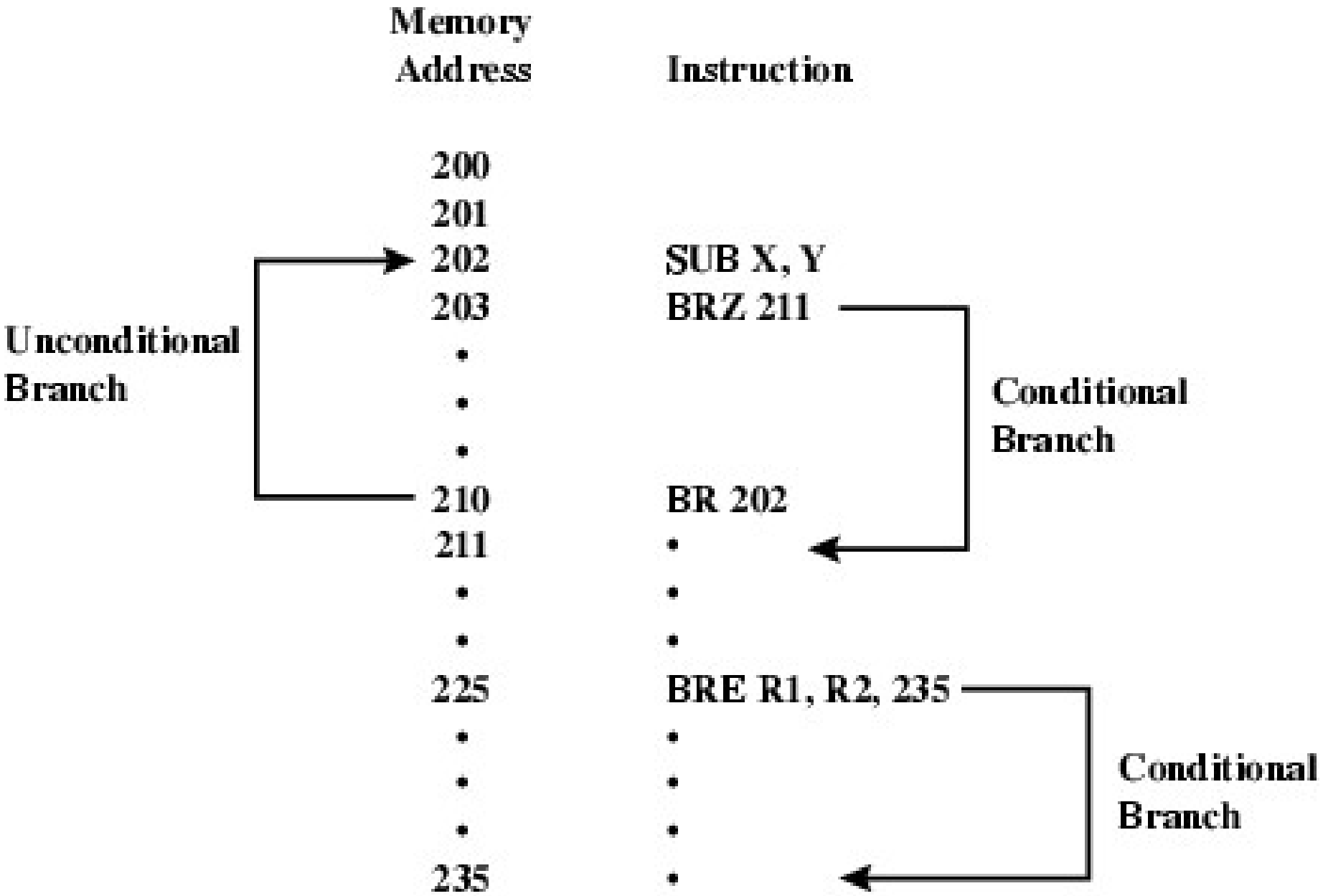
Transfer of Control

- Jump / Branch (Unconditional / Conditional)
  - e.g. jump to x if result is zero
- Skip (Unconditional / Conditional)
  - skip (unconditional) : Increment to skip next instruction
  - e.g. increment and skip if zero

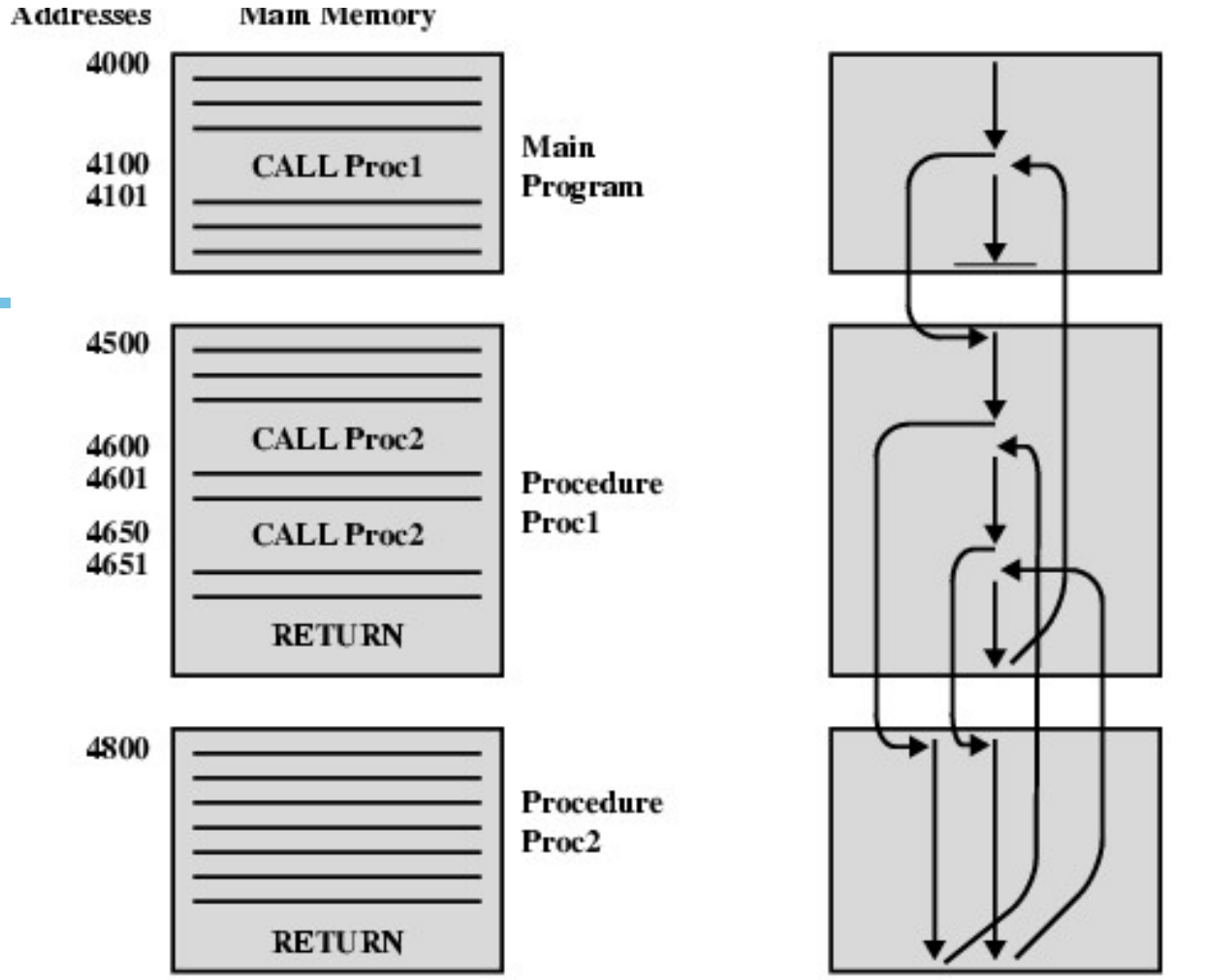
```
ISZ Register1
Branch xxxx
ADD A
```

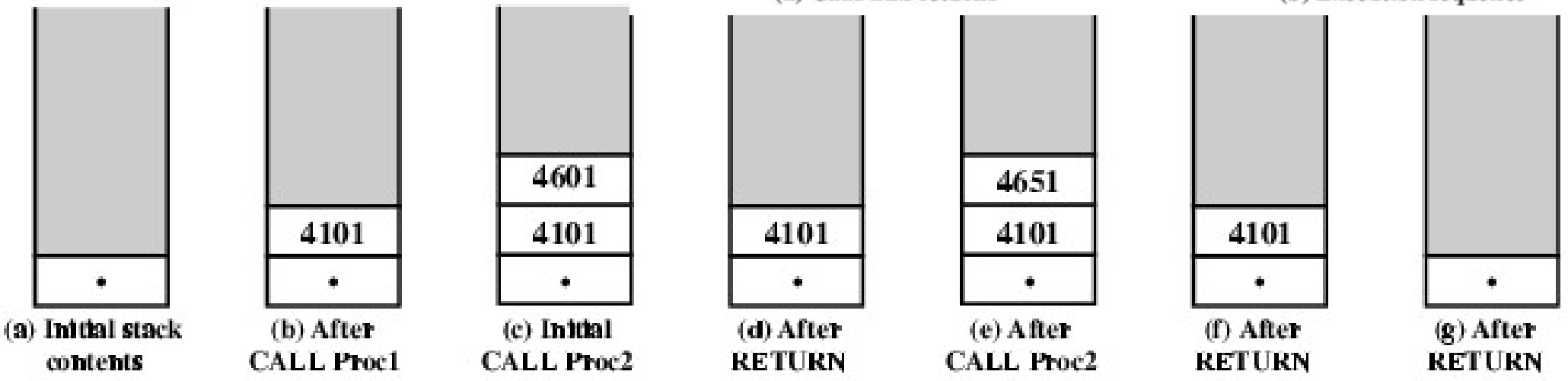
- Subroutine call
- interrupt call

Branch / Jump Instruction



Use of Stack


Addresses
Main Memory
4000
4100
4101
CALL Proc1
Main Program
4500
4600
4601
4650
4651
CALL Proc2
CALL Proc2
RETURN
Procedure Proc1
4800
RETURN
Procedure Proc2
(a) Calls and returns
(b) Execution sequence

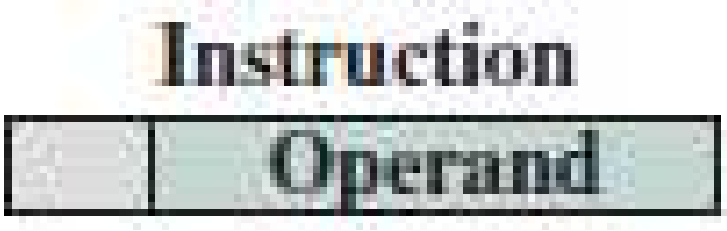

4101
4601
4101
4101
4651
4101
4101
(a) Initial stack contents
(b) After CALL Proc1
(c) Initial CALL Proc2
(d) After RETURN
(e) After CALL Proc2
(f) After RETURN
(g) After RETURN

Addressing Modes

- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Stack

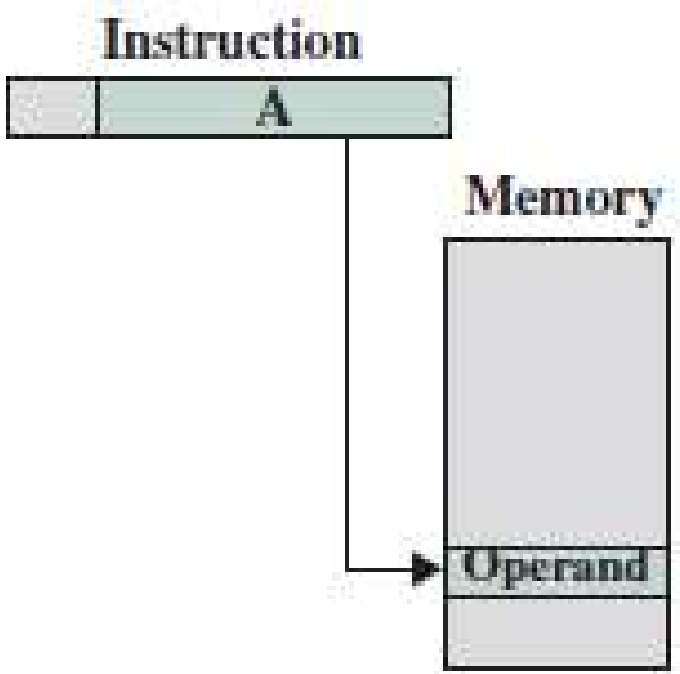
Immediate Addressing

- Operand is specified in the instruction itself
- e.g. ADD #5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



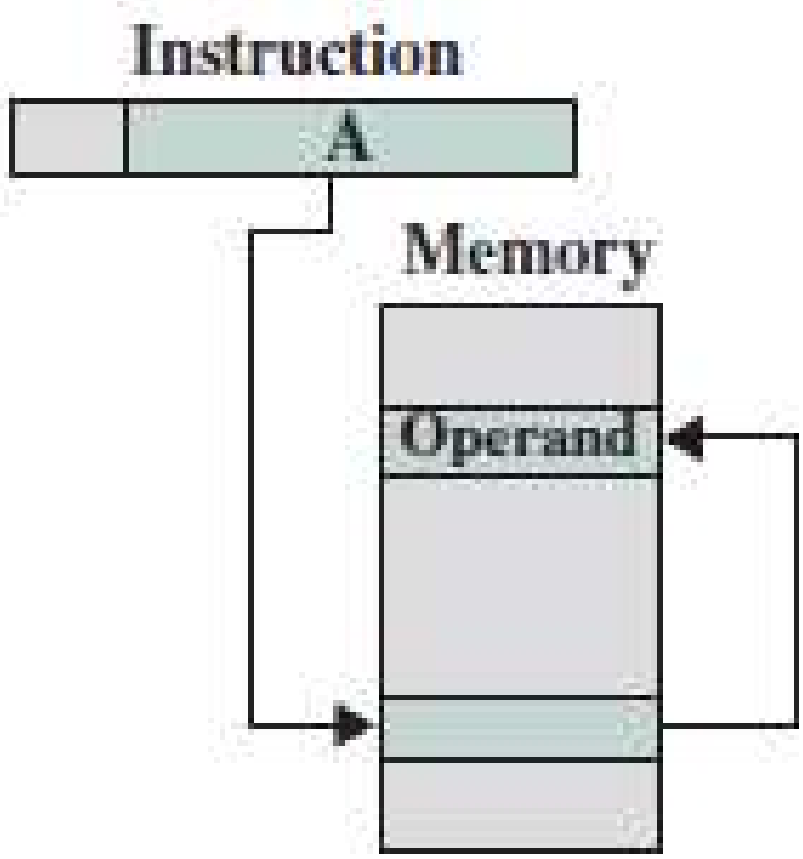
Direct Addressing

- Address of the operand is specified in the instruction
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of memory cell whose address is A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



Indirect Addressing

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

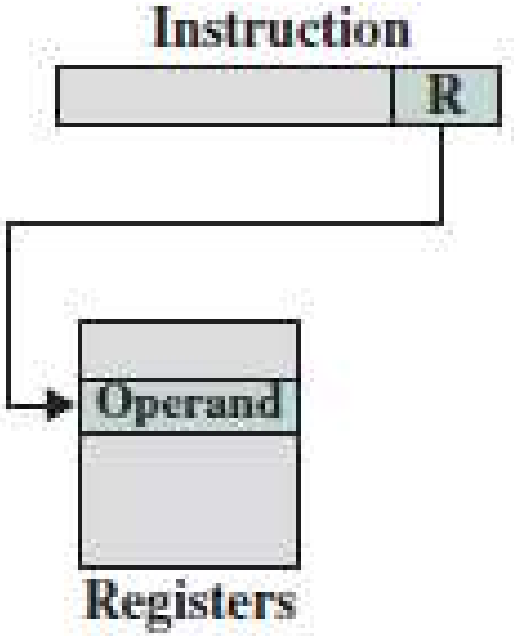


Indirect Addressing…

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

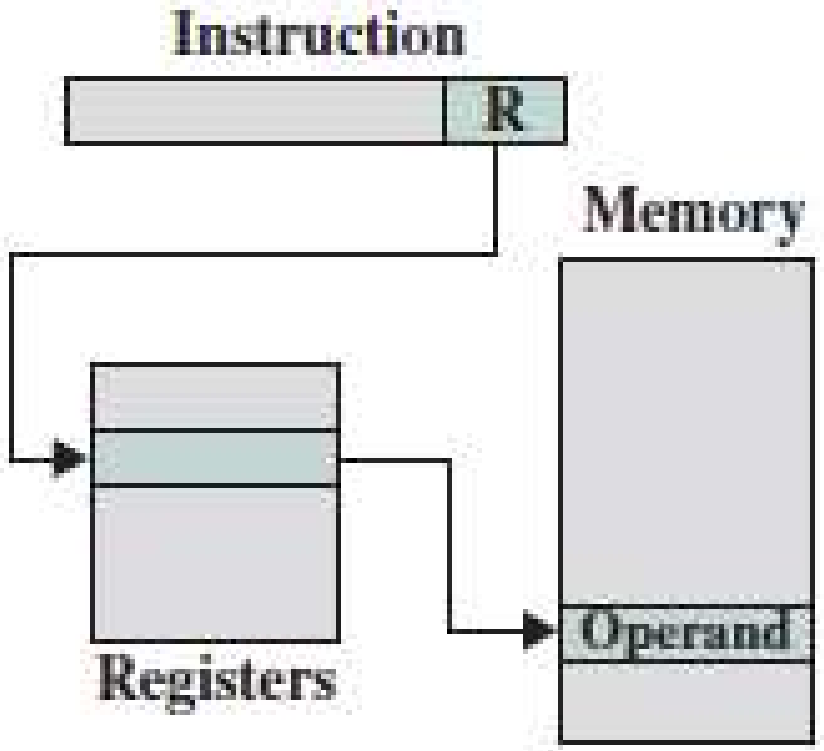
Register Addressing

- 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
- No memory access hence Very fast execution but very limited address space
- Multiple registers helps in improving performance
  - Requires good assembly programming or compiler writing
  - C programming : register int a;



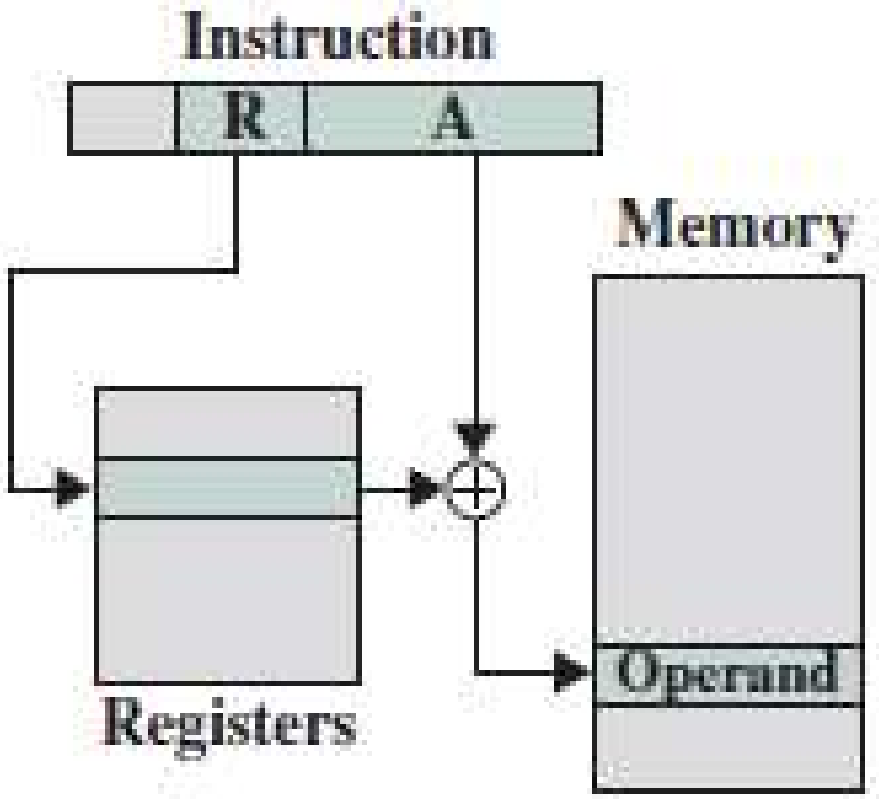
Register Indirect Addressing

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



Displacement Addressing

- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa
- Three variants:
  - Relative addressing
  - Base register addressing
  - Indexing



Relative Addressing

- Also known as PC relative addressing
- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- Relative addressing exploits the concept of locality

Base-Register Addressing

- The referenced register "R" contains a main memory address
- EA = R + A
- address field contains a displacement A
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexed Addressing

- The address field references a main memory address A
- The referenced register R contains a positive displacement from that address.
- EA = A + R
- Good for accessing arrays
  - EA = A + R
  - R++

Auto Indexing

- Auto indexing incase certain registers are devoted exclusively to indexing

  EA = A + (R)

  (R) ← (R) + 1

- Example: LODSB

  AL ← DS:[SI]

  SI is incremented or decremented based on direction flag.

  D = 0 → increment SI

  D = 1 → decrement SI

Post-indexing

- indexing is performed after the indirection

  EA = (A) + (R)

- Steps:
  1. The contents of the address field are used to access a memory location containing a direct address.
  2. Address is then indexed by the register value
- Use:
  - for accessing one of a number of blocks of data of a fixed format

Pre-indexing

innovate achieve lead

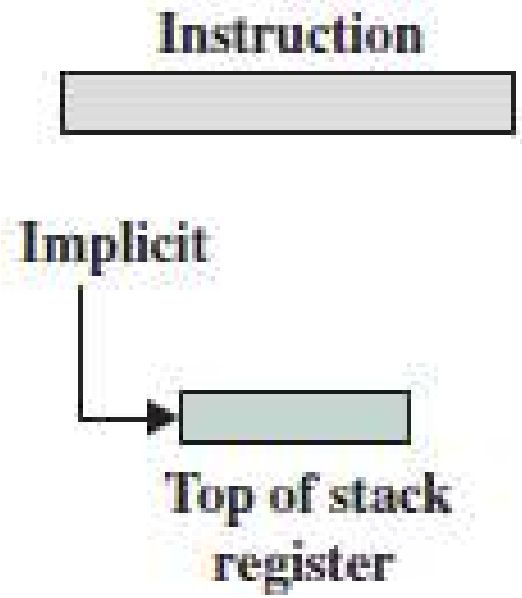
- An address is calculated as with simple indexing

  $EA = (A + (R))$

- Use:
  - to construct a multiway branch table

Stack Addressing

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



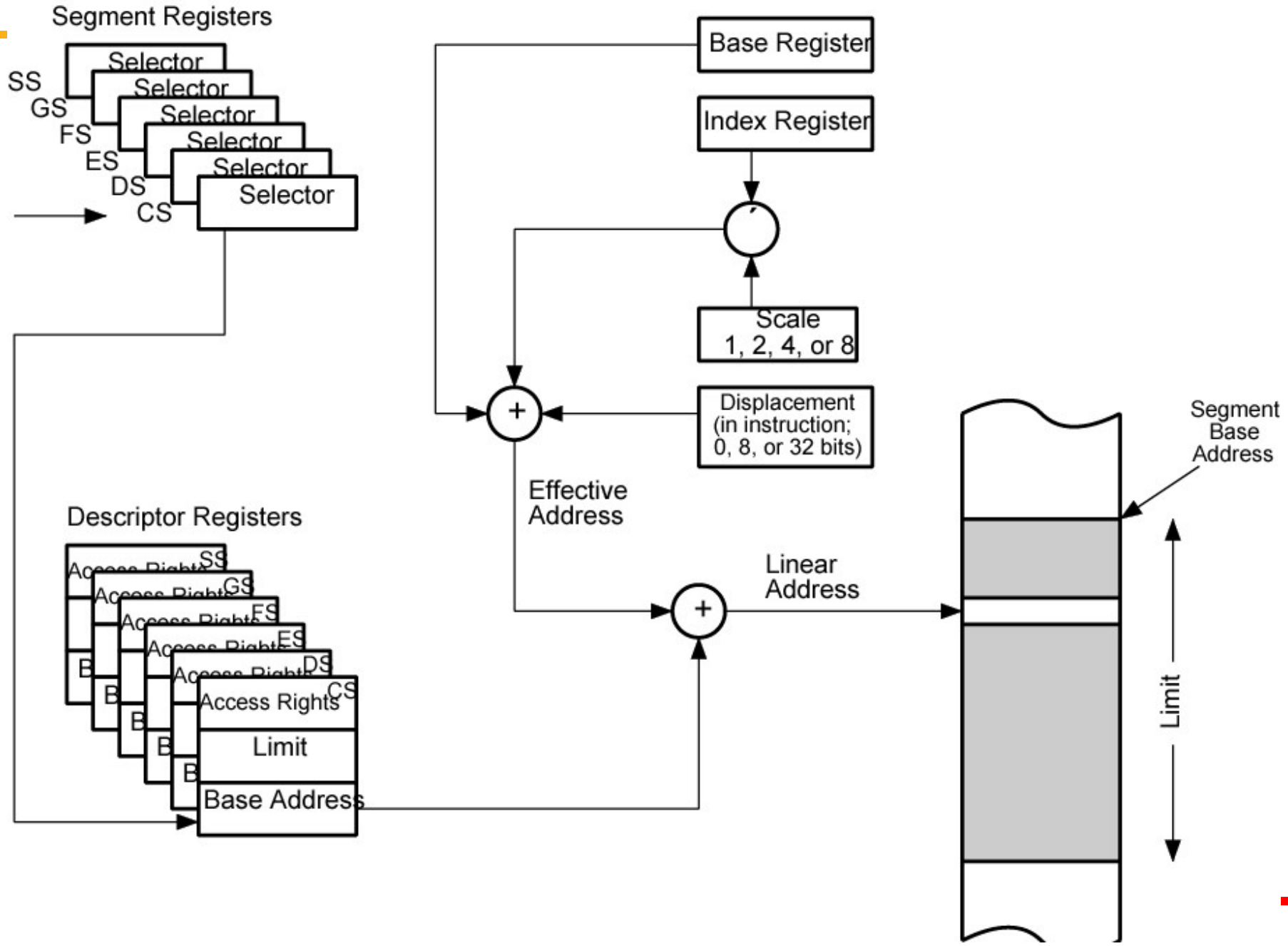
x86 Addressing Modes

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

32 bit registers ➔ EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
16 bit registers ➔ AX, BX, CX, DX, SI, DI, SP, BP),
8 bit registers ➔ AH, AL, BH, BL, CH, CL, DH , DL
Segment registers ➔ CS, DS, ES, SS, FS, GS

x86 Addressing Mode Calculation



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

Trade off between powerful instruction repertoire and saving space

Allocation of Bits

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