William Stallings
Computer Organization
and Architecture
8th Edition

Chapter 11

Instruction Sets:

Addressing Modes and Formats

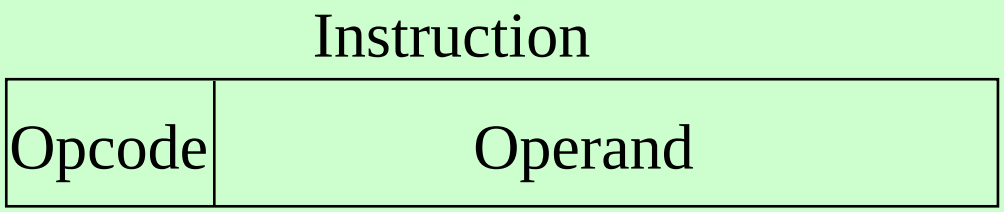
# Addressing Modes

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

# Immediate Addressing

- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

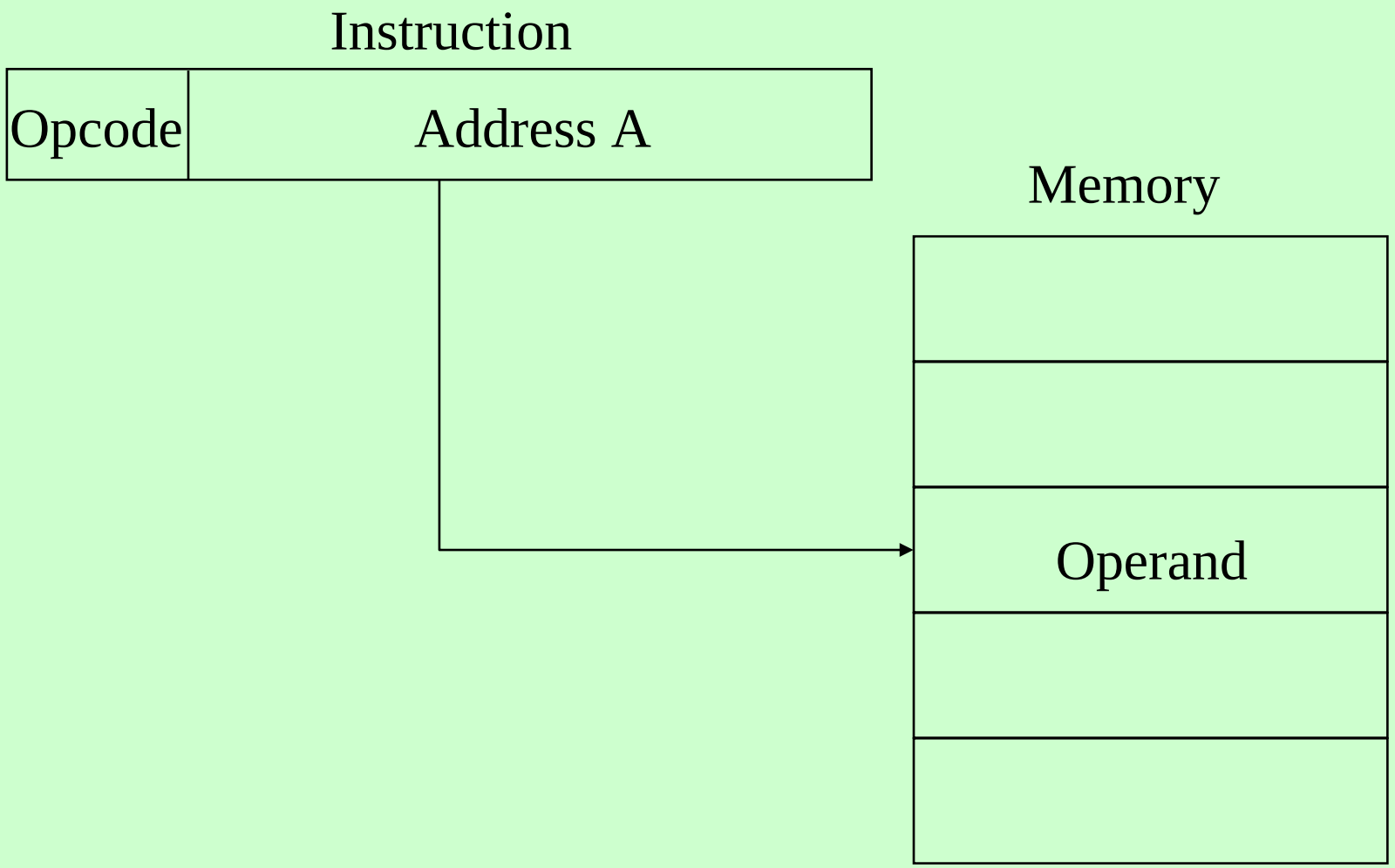
# Immediate Addressing Diagram



# Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g.  ADD A
  - Add contents of cell 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

# Direct Addressing Diagram



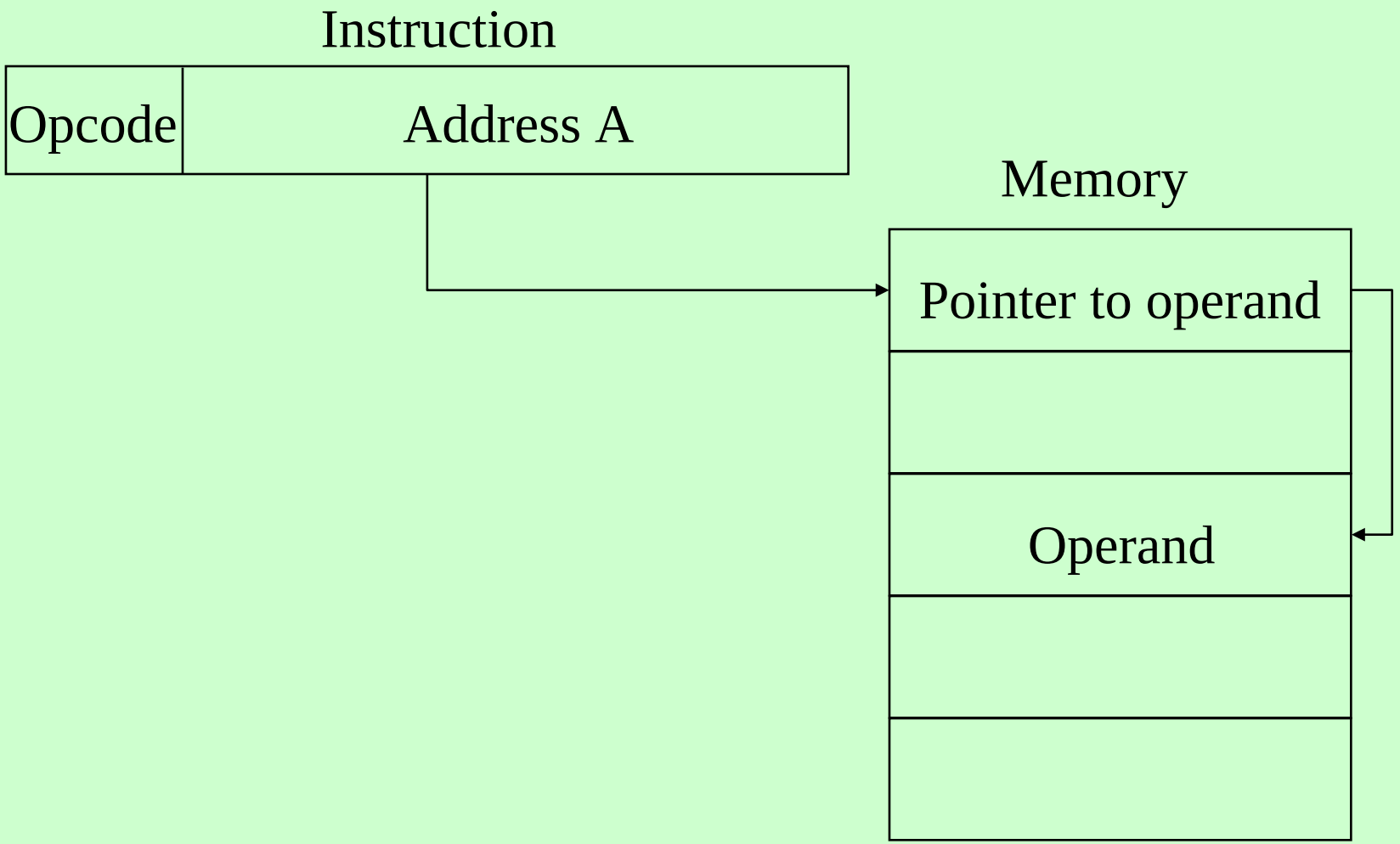
# Indirect Addressing (1)

- 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 (A)
  - Add contents of cell pointed to by contents of A to accumulator

# Indirect Addressing (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

# Indirect Addressing Diagram



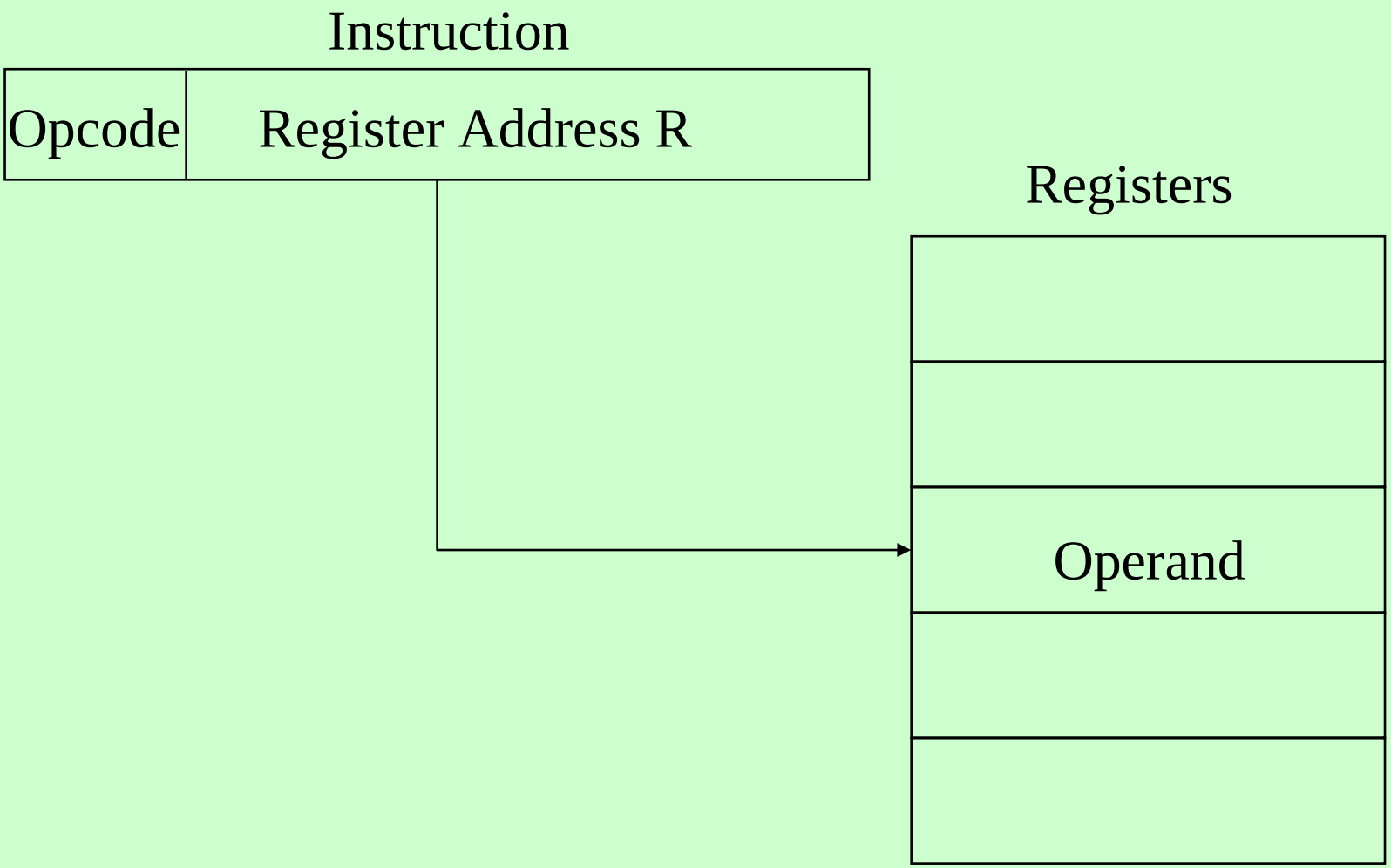
# Register Addressing (1)

- 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

# Register Addressing (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

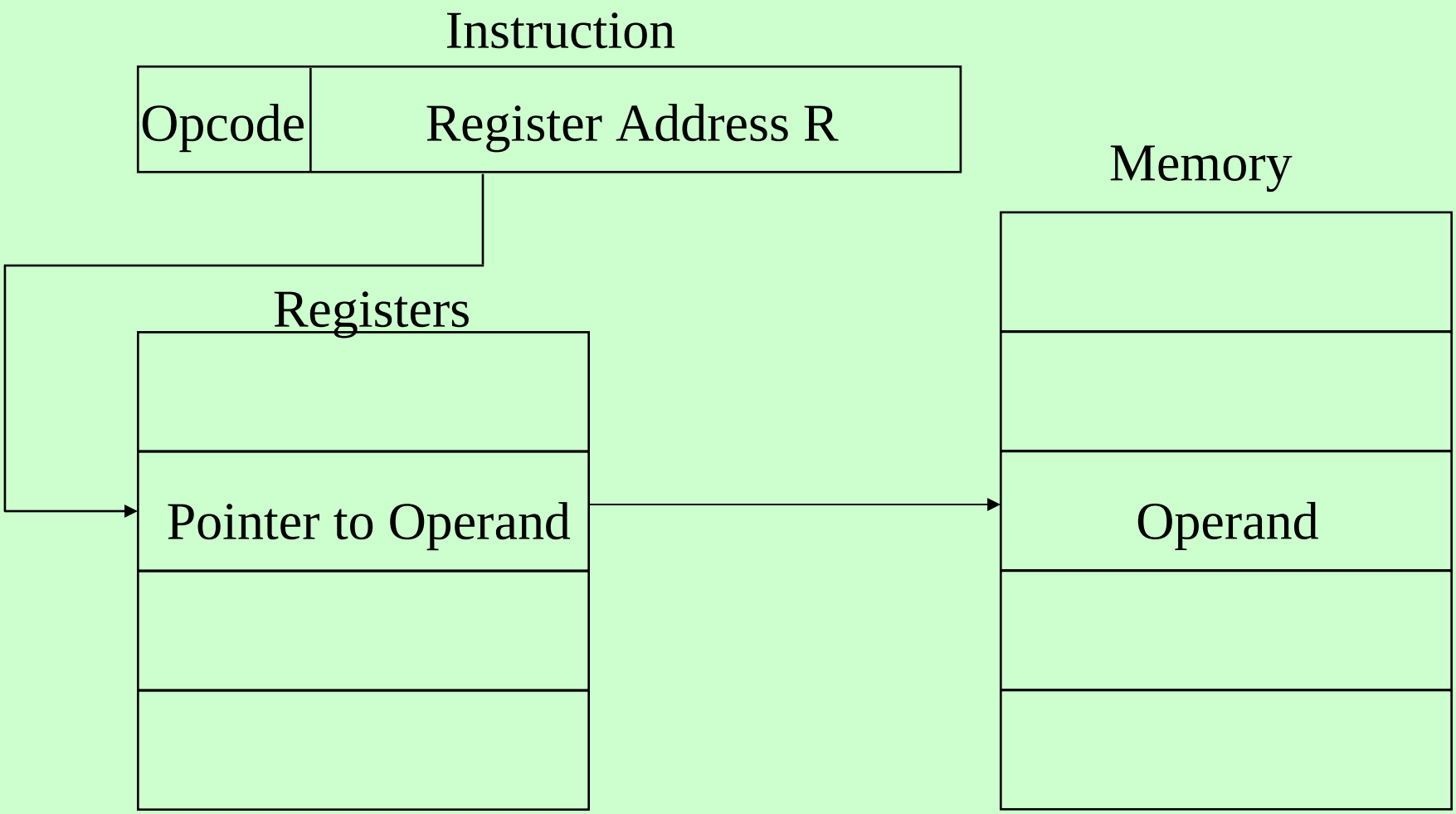
# Register Addressing Diagram



# Register Indirect Addressing

- C.f. 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

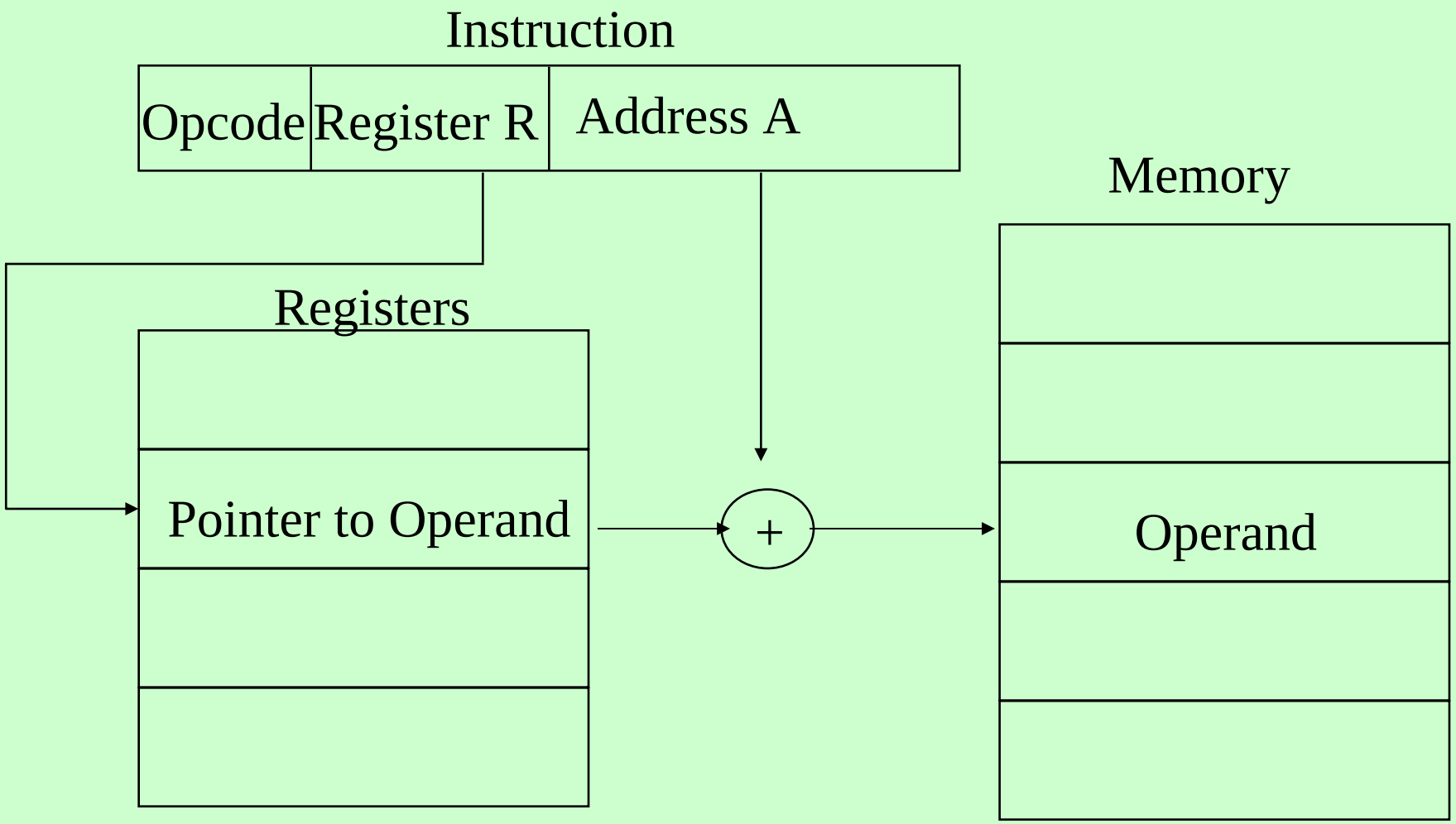
# Register Indirect Addressing Diagram



# Displacement Addressing

- EA = A + (R)
- Address field hold two values
  - —A = base value
  - —R = register that holds displacement
  - —or vice versa

# Displacement Addressing Diagram



# Relative Addressing

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC
- c.f locality of reference & cache usage

# Base-Register Addressing

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x86

# Indexed Addressing

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

# Combinations

- Postindex
- EA = (A) + (R)

- Preindex
- EA = (A+(R))

- (Draw the diagrams)

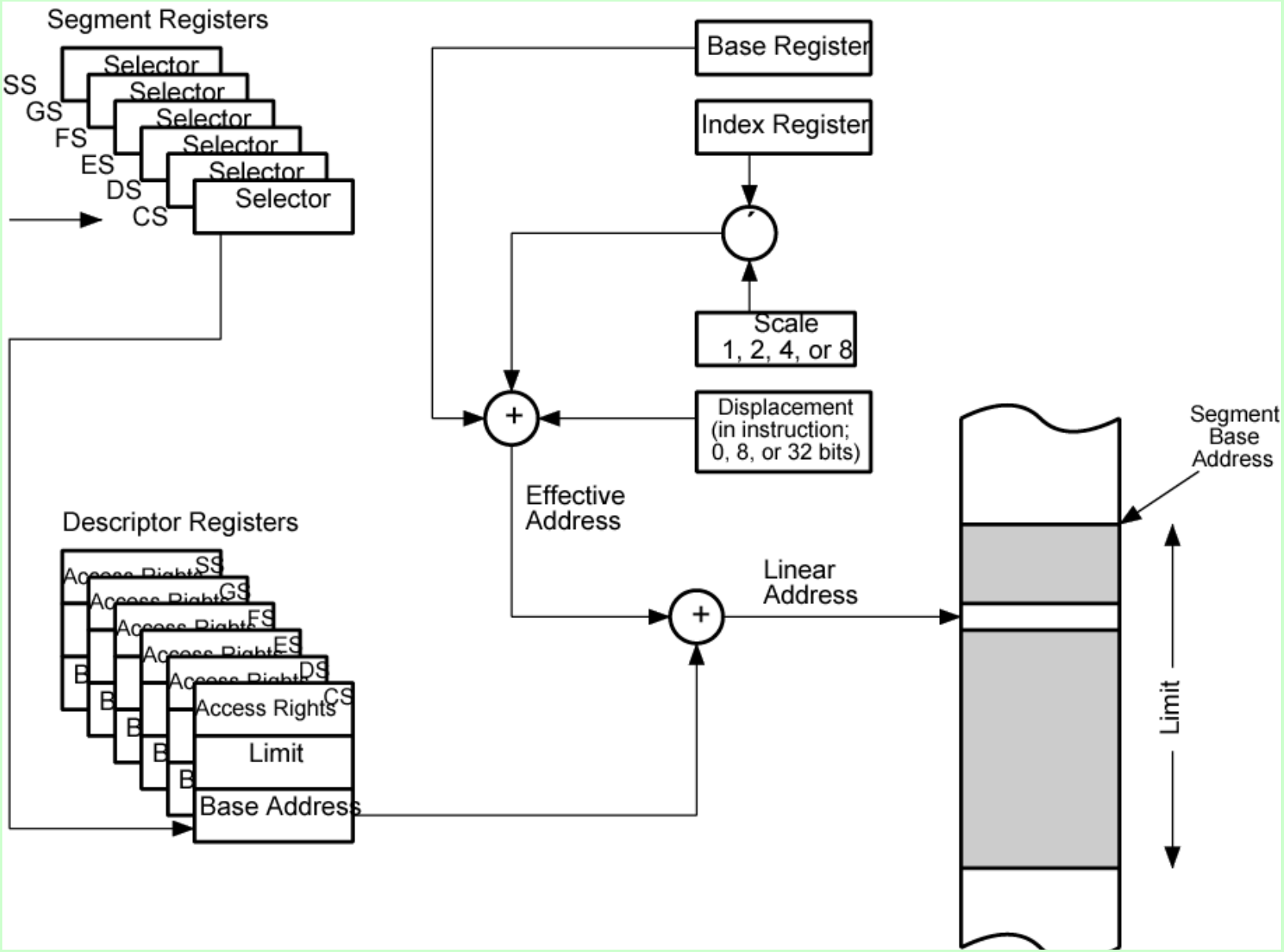
# Stack Addressing

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

# x86 Addressing Modes

- Virtual or effective address is offset into segment
  - Starting address plus offset gives linear address
  - This goes through page translation if paging enabled
- 12 addressing modes available
  - Immediate
  - Register operand
  - Displacement
  - Base
  - Base with displacement
  - Scaled index with displacement
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative

# x86 Addressing Mode Calculation

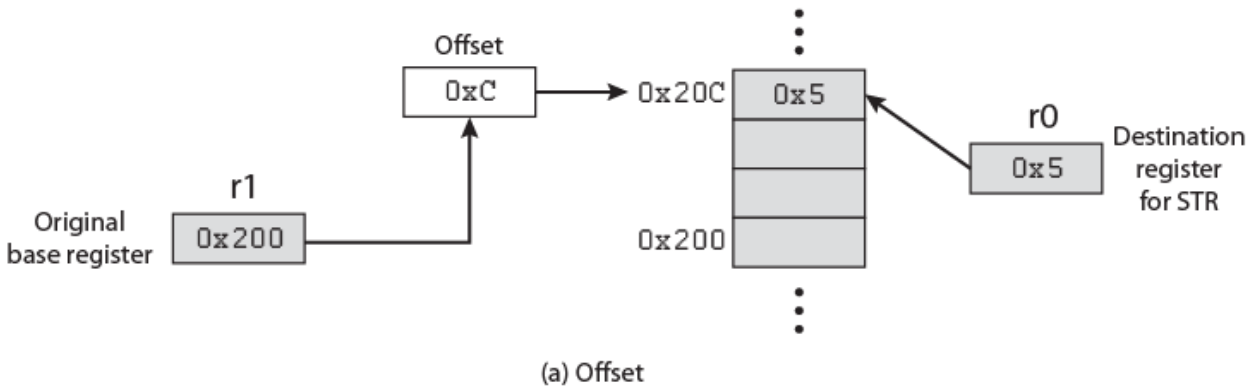


# ARM Addressing Modes
# Load/Store

- Only instructions that reference memory
- Indirectly through base register plus offset
- Offset
  - Offset added to or subtracted from base register contents to form the memory address
- Preindex
  - Memory address is formed as for offset addressing
  - Memory address also written back to base register
  - So base register value incremented or decremented by offset value
- Postindex
  - Memory address is base register value
  - Offset added or subtracted
    Result written back to base register
- Base register acts as index register for preindex and postindex addressing
- Offset either immediate value in instruction or another register
- If register scaled register addressing available
  - Offset register value scaled by shift operator
  - Instruction specifies shift size

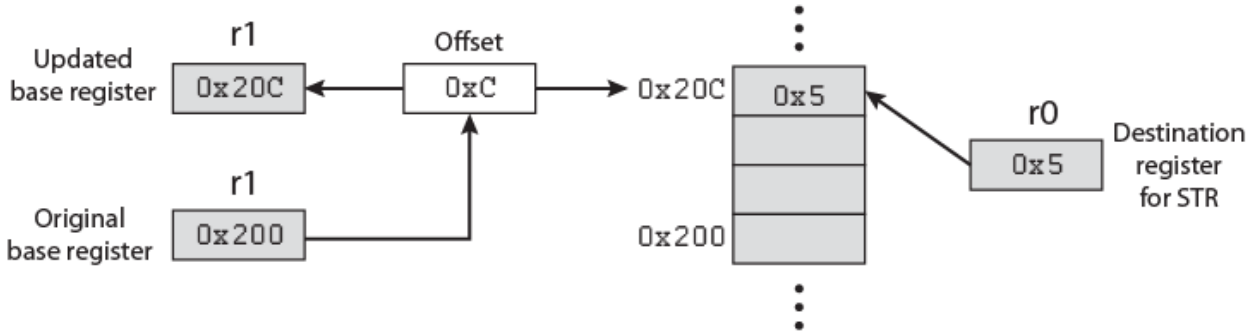
# ARM Indexing Methods

```
STRB r0, [r1, #12]
```



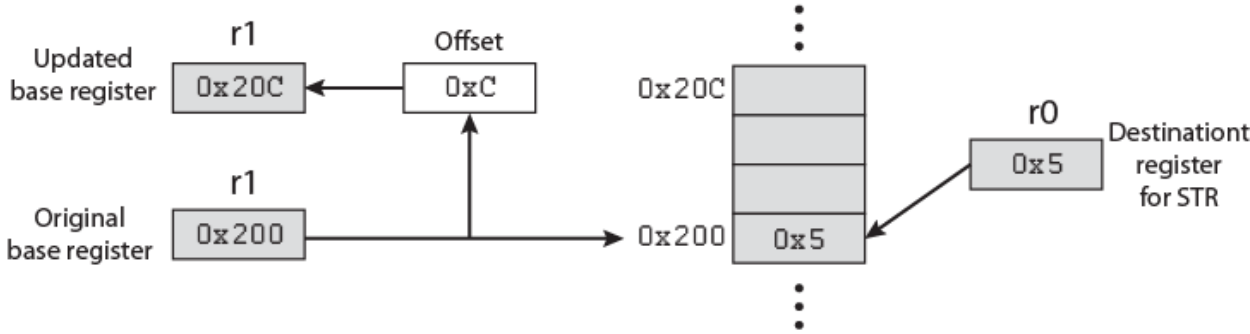

(a) Offset

```
STRB r0, [r1, #12]!
```




(b) Preindex

```
STRBv r0, [r1], #12
```




(c) Postindex

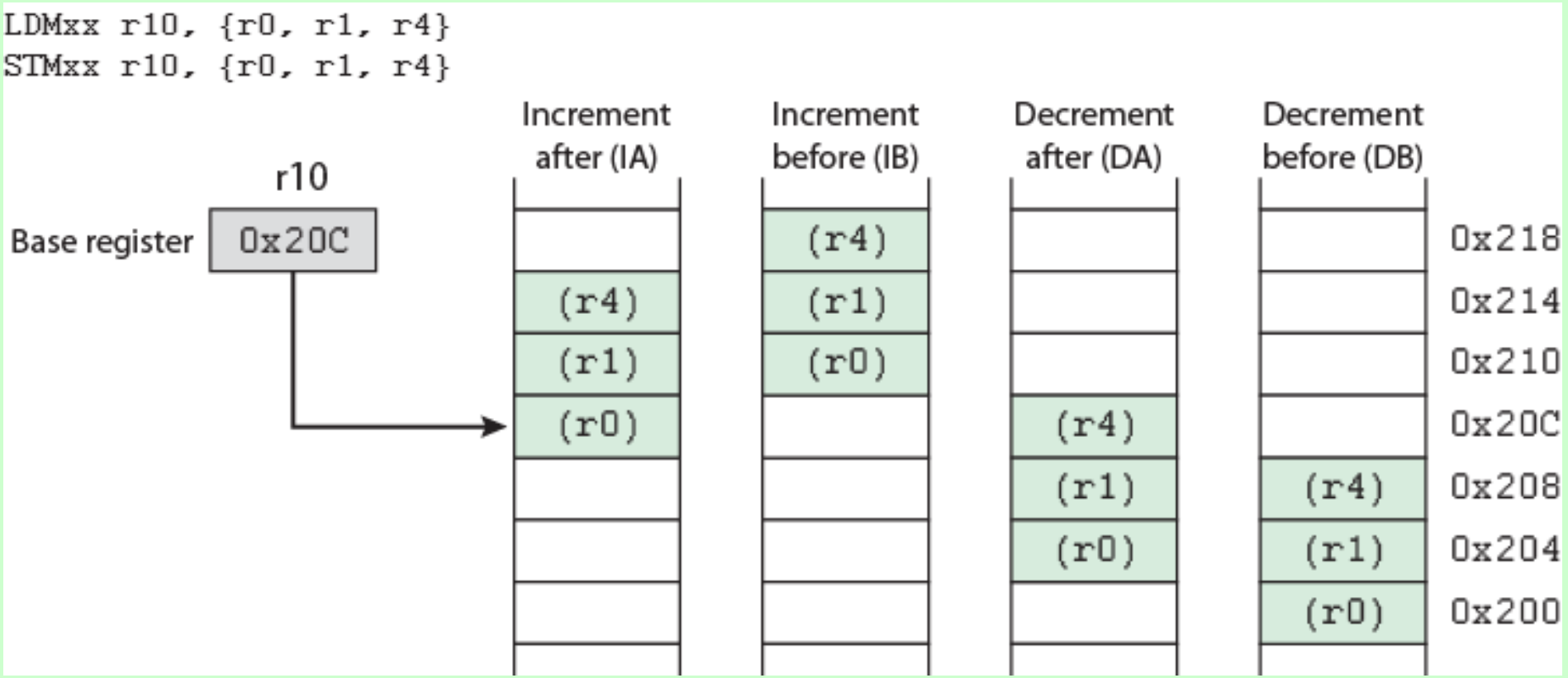
# ARM Data Processing Instruction Addressing & Branch Instructions

- Data Processing
  - —Register addressing
    - Value in register operands may be scaled using a shift operator
  - —Or mixture of register and immediate addressing
- Branch
  - —Immediate
  - —Instruction contains 24 bit value
  - —Shifted 2 bits left
    - On word boundary
    - Effective range +/-32MB from PC.

# ARM Load/Store Multiple Addressing

- Load/store subset of general-purpose registers
- 16-bit instruction field specifies list of registers
- Sequential range of memory addresses
- Increment after, increment before, decrement after, and decrement before
- Base register specifies main memory address
- Incrementing or decrementing starts before or after first memory access

# ARM Load/Store Multiple Addressing Diagram



# 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

# PDP-8 Instruction Format

**Memory Reference Instructions**

| Opcode | | | D/I | Z/C | Displacement | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 2 | 3 | 4 | 5 | | | | | | 11 |

**Input/Output Instructions**

| 1 | 1 | 0 | Device | | | | | | Opcode | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 2 | 3 | | | | | 8 | 9 | | 11 |

**Register Reference Instructions**

Group 1 Microinstructions

| 1 | 1 | 1 | 0 | CLA | CLL | CMA | CML | RAR | RAL | BSW | IAC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Group 2 Microinstructions

| 1 | 1 | 1 | 1 | CLA | SMA | SZA | SNL | RSS | OSR | HLT | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

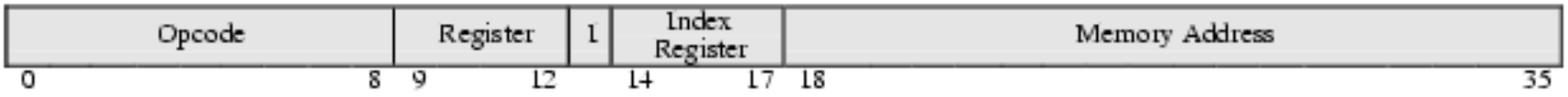
Group 3 Microinstructions

| 1 | 1 | 1 | 1 | CLA | MQA | 0 | MQL | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

D/I = Direct/Indirect address
Z/C = Page 0 or Current page
CLA = Clear Accumulator
CLL = Clear Link
CMA = CoMplement Accumulator
CML = CoMplement Link
RAR = Rotate Accumultator Right
RAL = Rotate Accumulator Left
BSW = Byte SWap

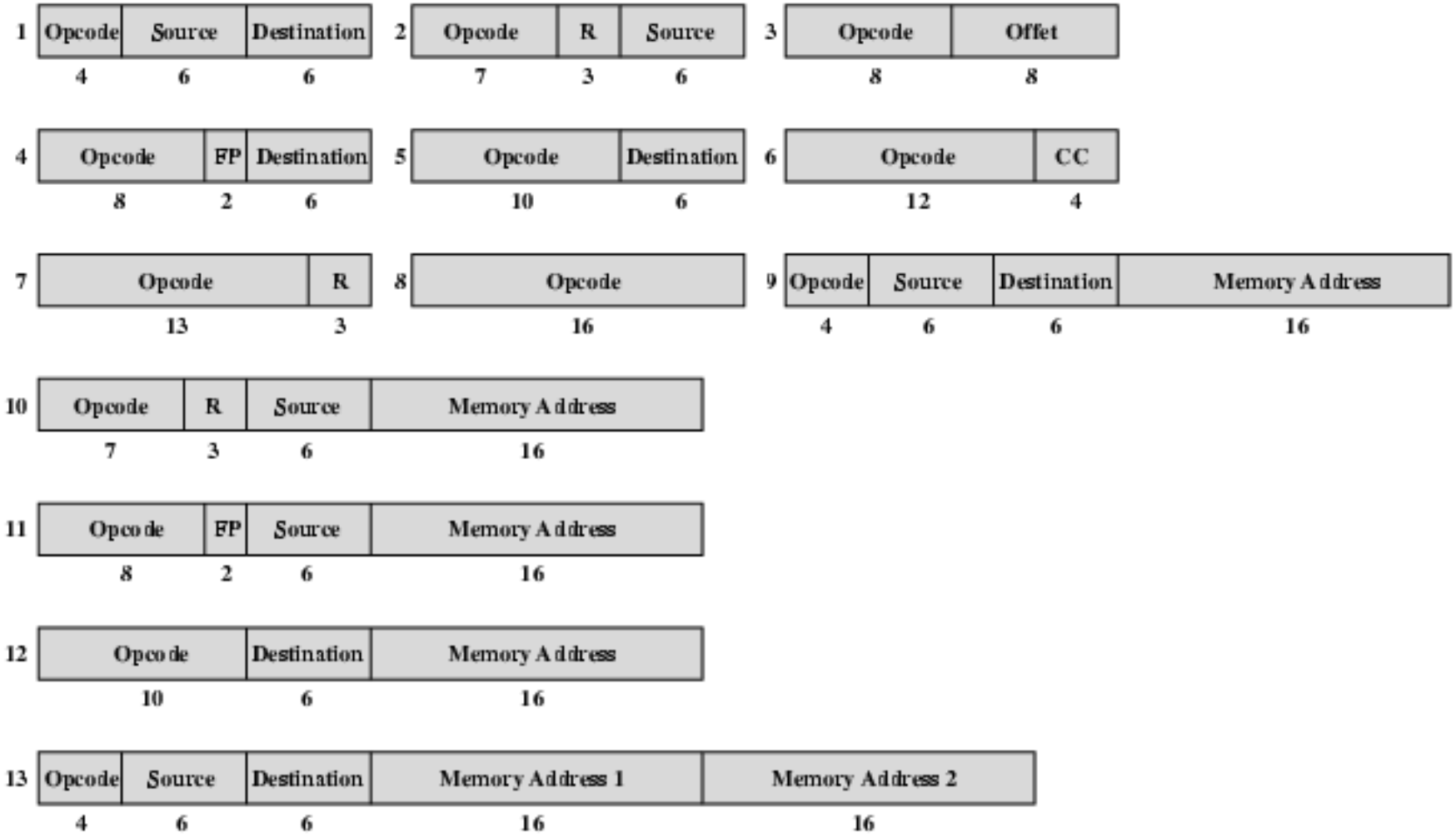
IAC = Increment ACcumulator
SMA = Skip on Minus Accumulator
SZA = Skip on Zero Accumulator
SNL = Skip on Nonzero Link
RSS = Reverse Skip Sense
OSR = Or with Switch Register
HLT = HaLT
MQA = Multiplier Quotient into Accumulator
MQL = Multiplier Quotient Load

# PDP-10 Instruction Format



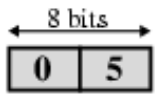
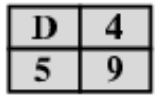
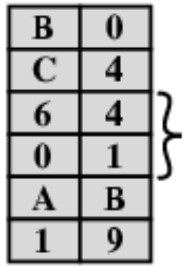
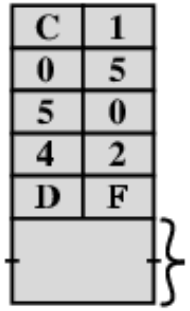

I = indirect bit

# PDP-11 Instruction Format

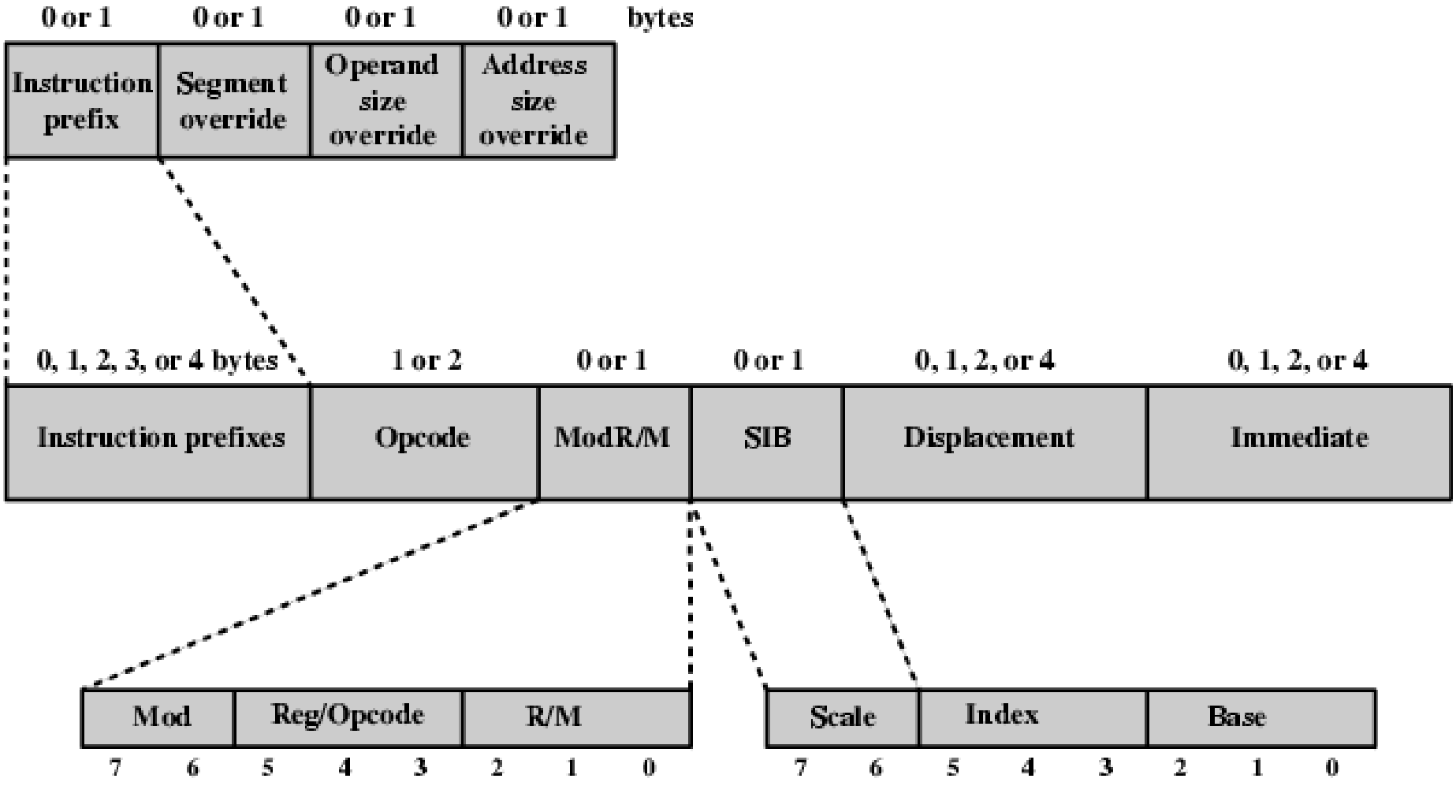



Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

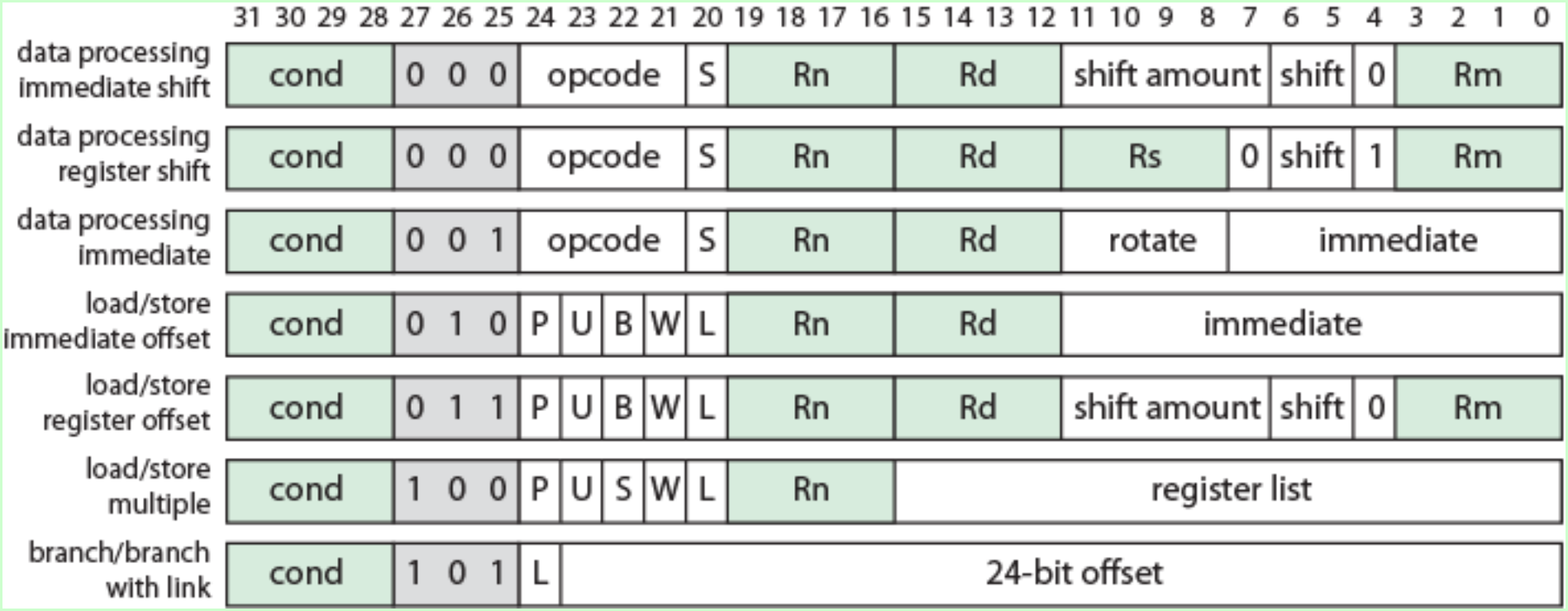
# VAX Instruction Examples

| Hexadecimal Format | Explanation | Assembler Notation and Description |
|---|---|---|
|   | Opcode for RSB | RSB<br>Return from subroutine |
|   | Opcode for CLRL<br>Register R9 | CLRL R9<br>Clear register R9 |
|   | Opcode for MOVW<br>Word displacement mode, Register R4<br>356 in hexadecimal<br>Byte displacement mode, Register R11<br>25 in hexadecimal | MOVW 356(R4), 25(R11)<br>Move a word from address that is 356 plus contents of R4 to address that is 25 plus contents of R11 |
|   | Opcode for ADDL3<br>Short literal 5<br>Register mode R0<br>Index prefix R2<br>Indirect word relative (displacement from PC)<br>Amount of displacement from PC relative to location A | ADDL3 #5, R0, @A[R2]<br>Add 5 to a 32-bit integer in R0 and store the result in location whose address is sum of A and 4 times the contents of R2 |

# x86 Instruction Format

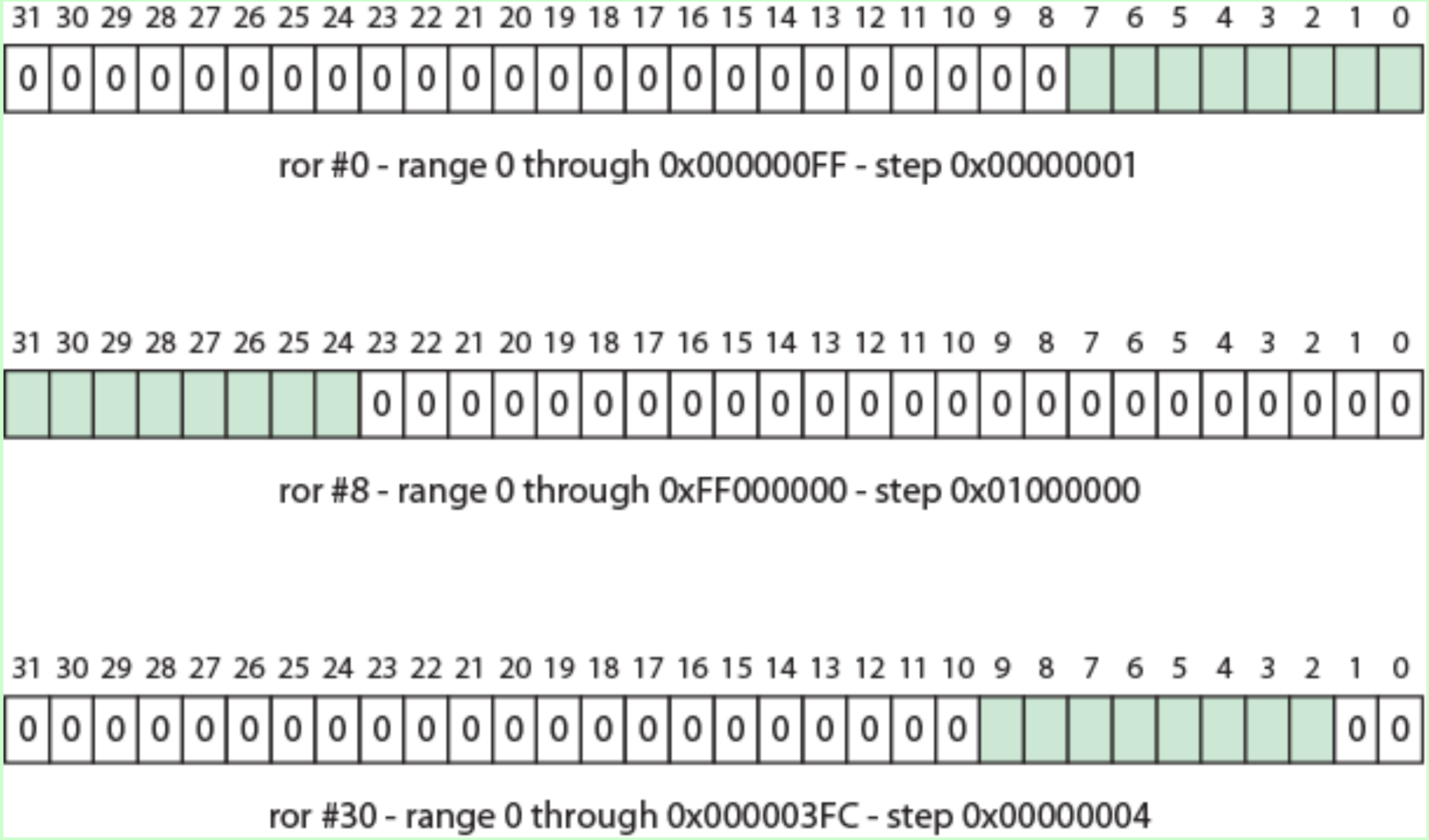


# ARM Instruction Formats




- S = For data processing instructions, updates condition codes
- S = For load/store multiple instructions, execution restricted to supervisor mode
- P, U, W = distinguish between different types of addressing_mode
- B = Unsigned byte (B==1) or word (B==0) access
- L = For load/store instructions, Load (L==1) or Store (L==0)
- L = For branch instructions, is return address stored in link register

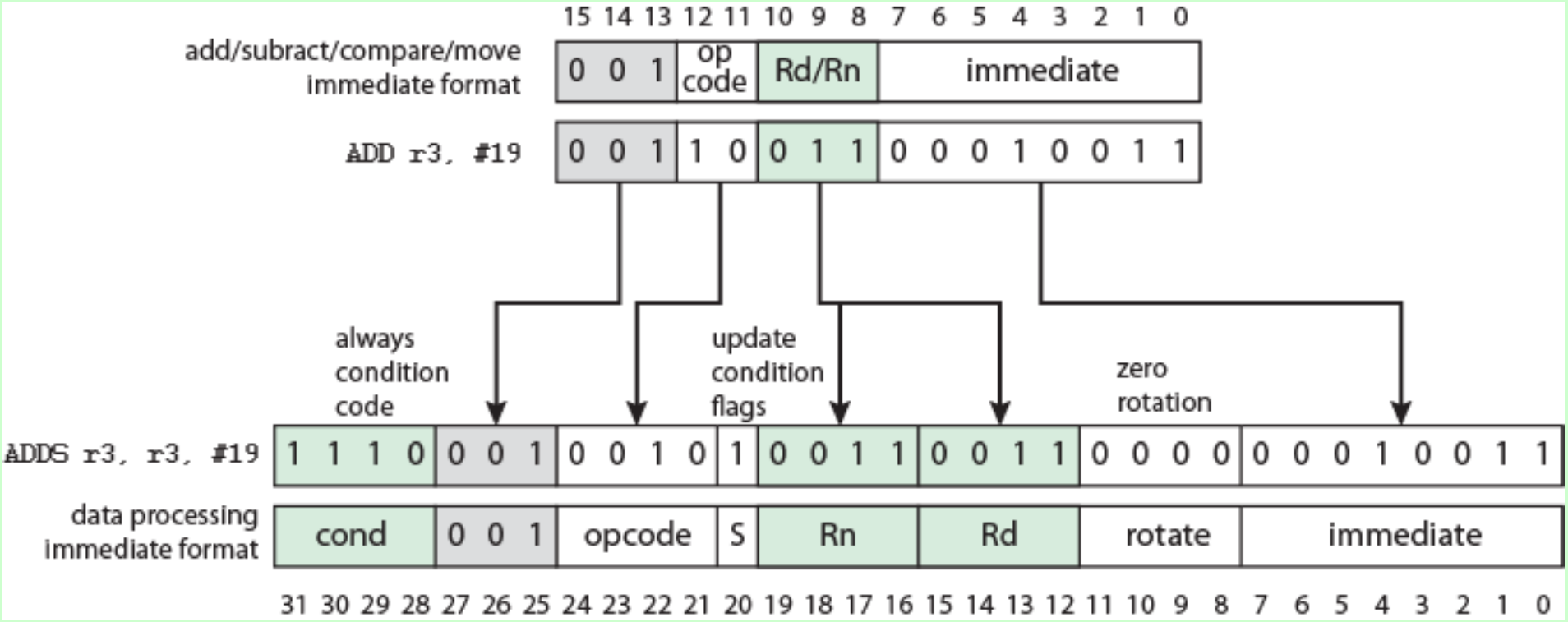
# ARM Immediate Constants Fig 11.11



# Thumb Instruction Set

- Re-encoded subset of ARM instruction set
- Increases performance in 16-bit or less data bus
- Unconditional (4 bits saved)
- Always update conditional flags
  - —Update flag not used (1 bit saved)
- Subset of instructions
  - —2 bit opcode, 3 bit type field (1 bit saved)
  - —Reduced operand specifications (9 bits saved)

# Expanding Thumb ADD Instruction to ARM Equivalent Fig 11.12



# Assembler

- Machines store and understand binary instructions
- E.g. N= I + J + K  initialize I=2, J=3, K=4
- Program starts in location 101
- Data starting 201
- Code:
- Load contents of 201 into AC
- Add contents of 202 to AC
- Add contents of 203 to AC
- Store contents of AC to 204
- Tedious and error prone

# Improvements

- Use hexadecimal rather than binary
  - Code as series of lines
    - Hex address and memory address
  - Need to translate automatically using program
- Add symbolic names or mnemonics for instructions
- Three fields per line
  - Location address
  - Three letter opcode
  - If memory reference: address
- Need more complex translation program

Program in:
Binary

| Address | Contents | | | |
|---|---|---|---|---|
| 101 | 0010 | 0010 | 101 | 2201 |
| 102 | 0001 | 0010 | 102 | 1202 |
| 103 | 0001 | 0010 | 103 | 1203 |
| 104 | 0011 | 0010 | 104 | 3204 |
| 201 | 0000 | 0000 | 201 | 0002 |
| 202 | 0000 | 0000 | 202 | 0003 |
| 203 | 0000 | 0000 | 203 | 0004 |
| 204 | 0000 | 0000 | 204 | 0000 |

Hexadecimal

| Address | Contents |
|---|---|
| 101 | 2201 |
| 102 | 1202 |
| 103 | 1203 |
| 104 | 3204 |
| 201 | 0002 |
| 202 | 0003 |
| 203 | 0004 |
| 204 | 0000 |

# Symbolic Addresses

- First field (address) now symbolic
- Memory references in third field now symbolic
- Now have assembly language and need an assembler to translate
- Assembler used for some systems programming
  - —Compliers
  - —I/O routines

# Symbolic Program

| Address | Instruction | |
|---|---|---|
| 101 | LDA | 201 |
| 102 | ADD | 202 |
| 103 | ADD | 203 |
| 104 | STA | 204 |
| | | |
| | | |
| 201 | DAT | 2 |
| 202 | DAT | 3 |
| 203 | DAT | 4 |
| 204 | DAT | 0 |

# Assembler Program

| Label | Operation | Operand |
|---|---|---|
| FORMUL | LDA | I |
| | ADD | J |
| | ADD | K |
| | STA | N |
| I | DATA | 2 |
| J | DATA | 3 |
| K | DATA | 4 |
| N | DATA | 0 |

# Foreground Reading

- Stallings chapter 11
- Intel and ARM Web sites