(Advanced) Computer Architechture

Prof. Dr. Hasan Hüseyin BALIK

(6th Week)

Outline

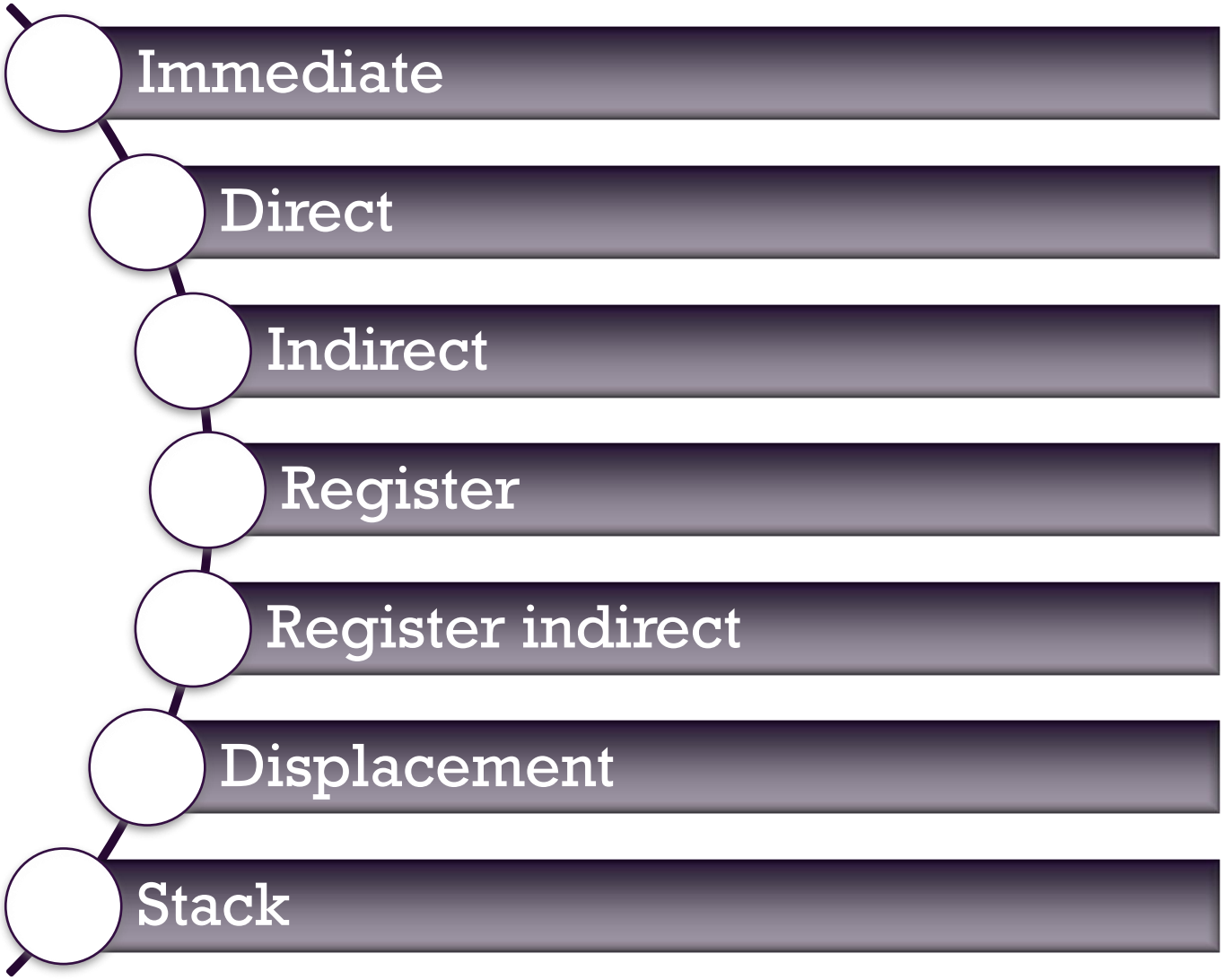
3. Instruction sets
   - —Instruction Sets: Characteristics and Functions
   - —Instruction Sets: Addressing Modes and Formats
   - —Assembly Language and Related Topics

3.2 Instruction Sets: Addressing Modes and Formats

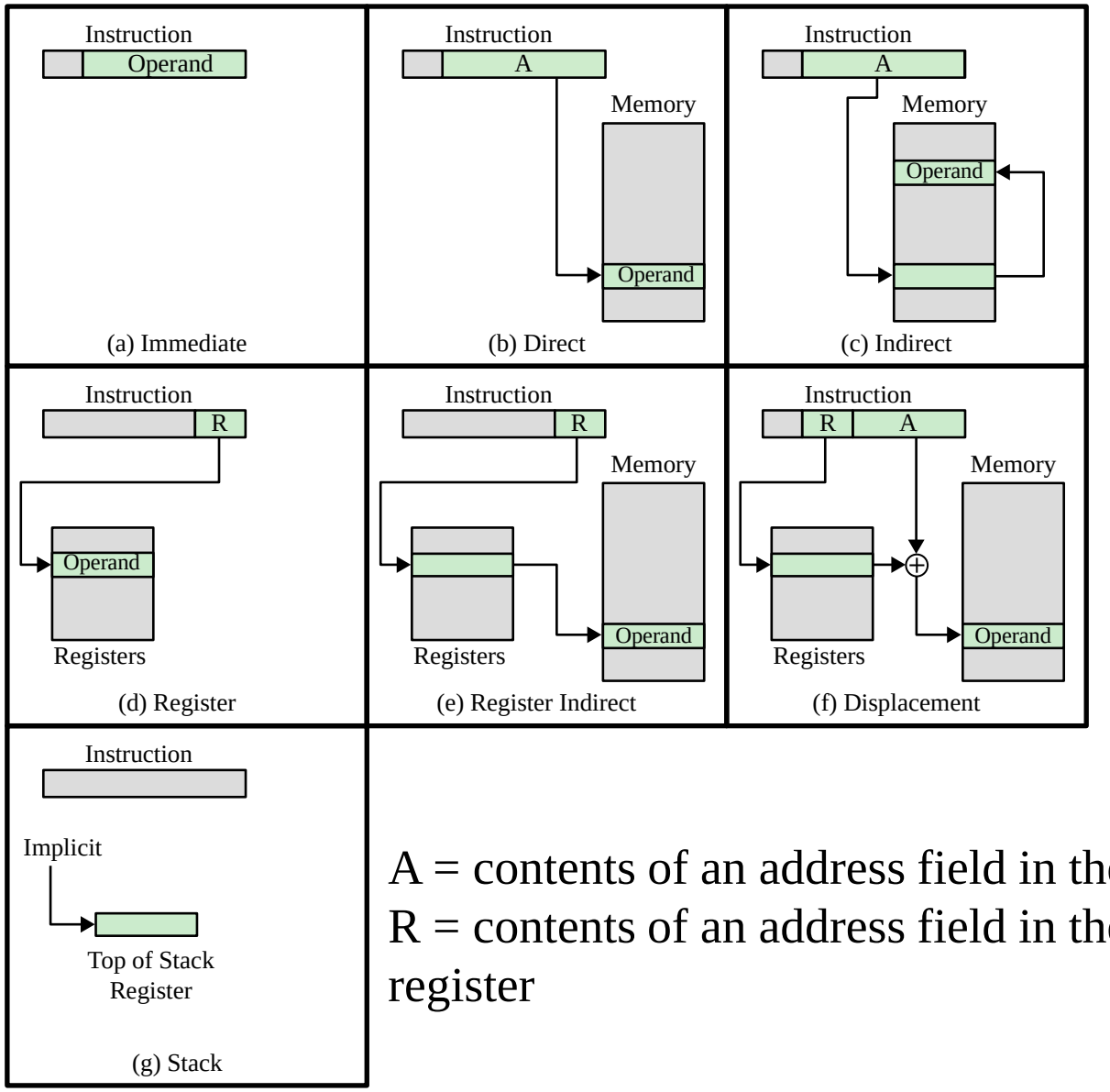
3.2 Outline

- Addressing Modes
- x86 and ARM Addressing Modes
- Instruction Formats
- x86 and ARM Instruction Formats

Addressing Modes



Addressing Modes



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

Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

A = contents of an address field in the instruction
R = contents of an address field in the 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

Immediate Addressing

- Simplest form of addressing
- Operand = A
- This mode can be used to define and use constants or set initial values of variables
  - Typically the number will be stored in twos complement form
  - The leftmost bit of the operand field is used as a sign bit
- Advantage:
  - 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
- Disadvantage:
  - 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

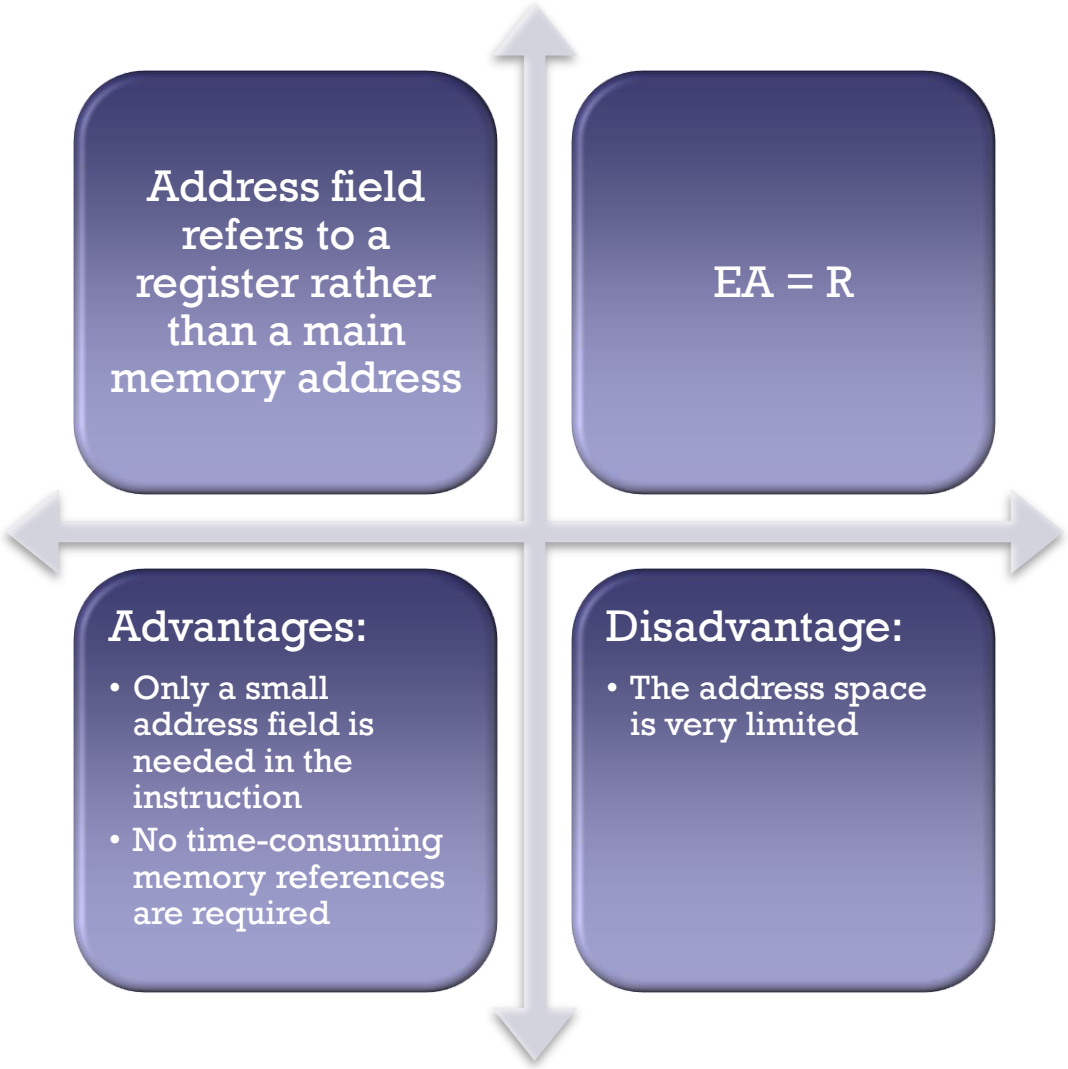
Direct Addressing

- Address field contains the effective address of the operand
- Effective address (EA) = address field (A)
- Was common in earlier generations of computers
- Requires only one memory reference and no special calculation
- Limitation is that it provides only a limited address space

Indirect Addressing

- Reference to the address of a word in memory which contains a full-length address of the operand
- EA = (A)
  - Parentheses are to be interpreted as meaning *contents of*
- 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 the operand
    - One to get its address and a second to get its value
- A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing
  - EA = (. . . (A) . . .)
  - Disadvantage is that three or more memory references could be required to fetch an operand

Register Addressing



Register Indirect Addressing

- Analogous to indirect addressing
  - The only difference is whether the address field refers to a memory location or a register
- EA = (R)
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than indirect addressing

Displacement Addressing

- Combines the capabilities of direct addressing and register indirect addressing
- EA = A + (R)
- Requires that the instruction have two address fields, at least one of which is explicit
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the effective address
- Most common uses:
  - Relative addressing
  - Base-register addressing
  - Indexing

Relative Addressing

The implicitly referenced register is the program counter (PC)

- The next instruction address is added to the address field to produce the EA
- Typically the address field is treated as a twos complement number for this operation
- Thus the effective address is a displacement relative to the address of the instruction

Exploits the concept of locality

Saves address bits in the instruction if most memory references are relatively near to the instruction being executed

Base-Register Addressing

- The referenced register contains a main memory address and the address field contains a displacement from that address
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing segmentation
- In some implementations a single segment base register is employed and is used implicitly
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it explicitly

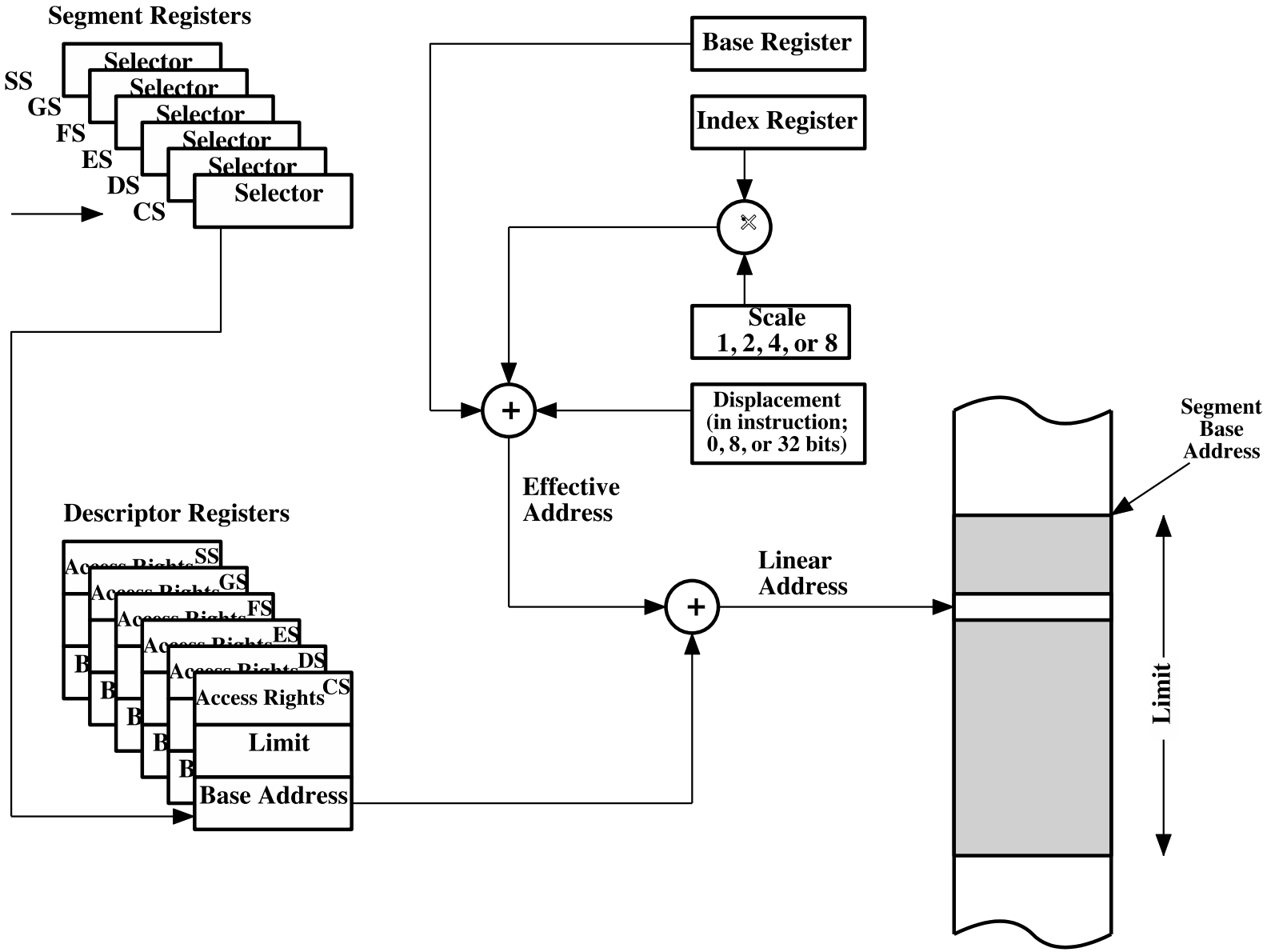
Indexing

- The address field references a main memory address and the referenced register contains a positive displacement from that address
- The method of calculating the EA is the same as for base-register addressing
- An important use is to provide an efficient mechanism for performing iterative operations
- Autoindexing
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (R)
  - (R) = (R) + 1
- Postindexing
  - Indexing is performed after the indirection
  - EA = (A) + (R)
- Preindexing
  - Indexing is performed before the indirection
  - EA = (A + (R))

Stack Addressing

- A stack is a linear array of locations
  - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
  - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
  - The stack pointer is maintained in a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack

x86 Addressing Mode Calculation



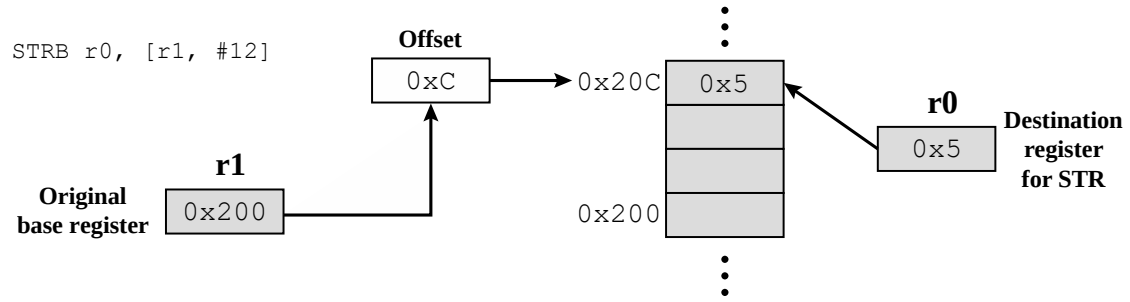
x86 Addressing Modes

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

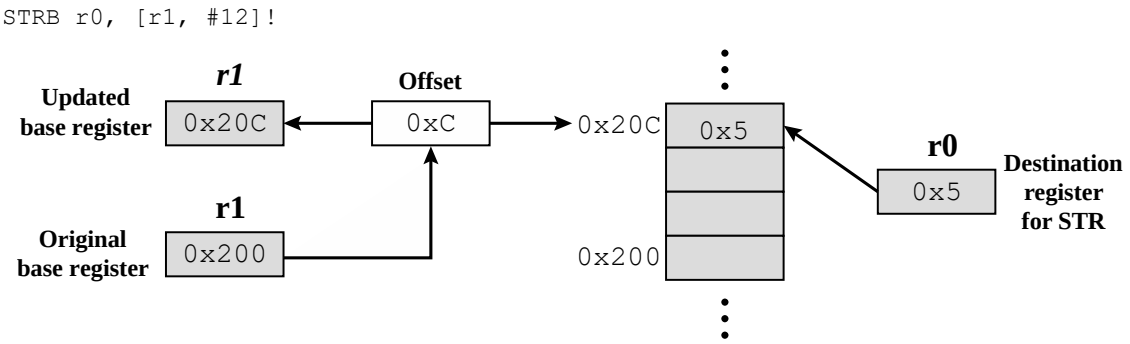
LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction

R = register
B = base register
I = index register
S = scaling factor

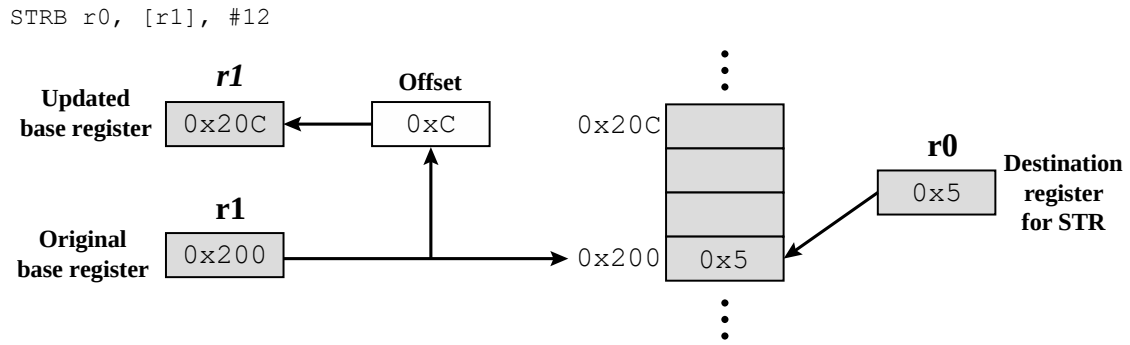
ARM Indexing Methods



(a) Offset



(b) Preindex



(c) Postindex

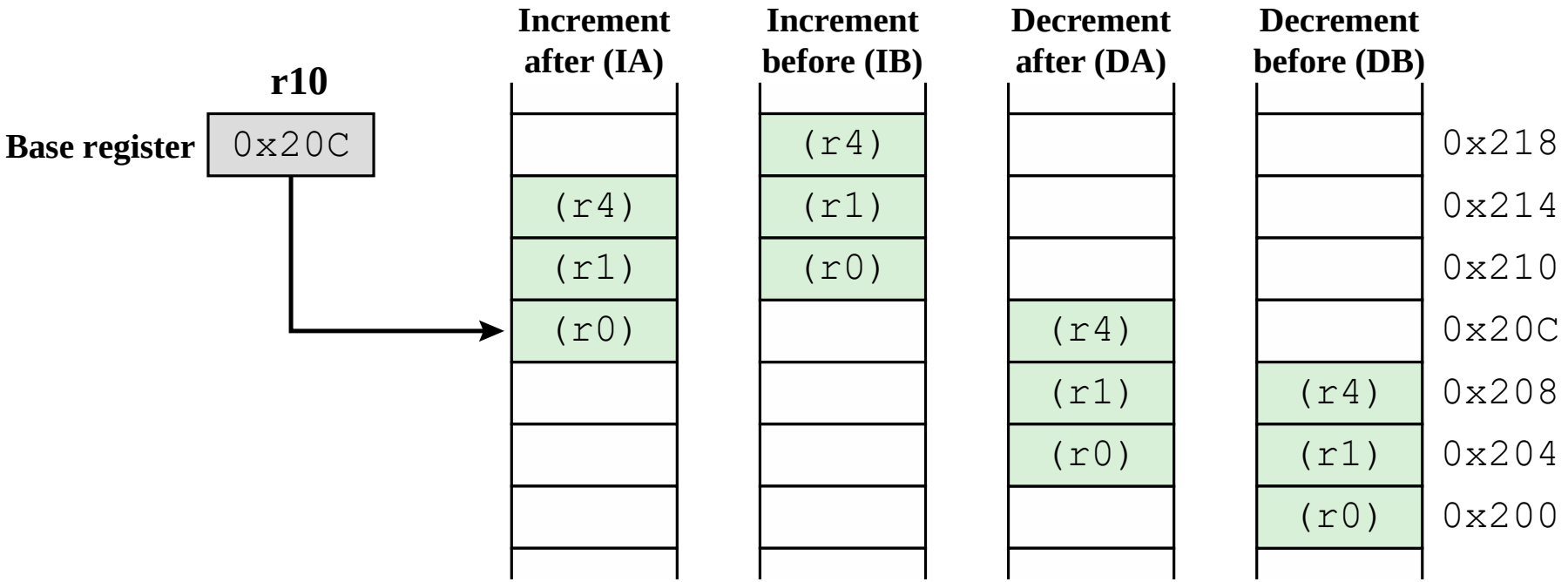
ARM Data Processing Instruction Addressing and Branch Instructions

- Data processing instructions
  - Use either register addressing or a mixture of register and immediate addressing
  - For register addressing the value in one of the register operands may be scaled using one of the five shift operators

- Branch instructions
  - The only form of addressing for branch instructions is immediate
  - Instruction contains 24 bit value
    - Shifted 2 bits left so that the address is on a word boundary
    - Effective range ± 32MB from from the program counter

ARM Load/Store Multiple Addressing

```
LDMxx r10, {r0, r1, r4}
STMxx r10, {r0, r1, r4}
```



Instruction Formats

Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

Instruction Length

- Most basic design issue
- Affects, and is affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- Should be equal to the memory-transfer length or one should be a multiple of the other
- Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers

Allocation of Bits

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

PDP-8 Instruction Formats

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

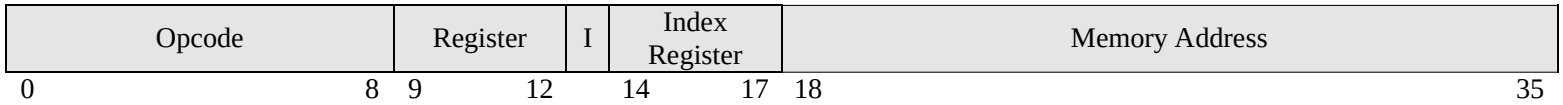
Group 3 Microinstructions

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

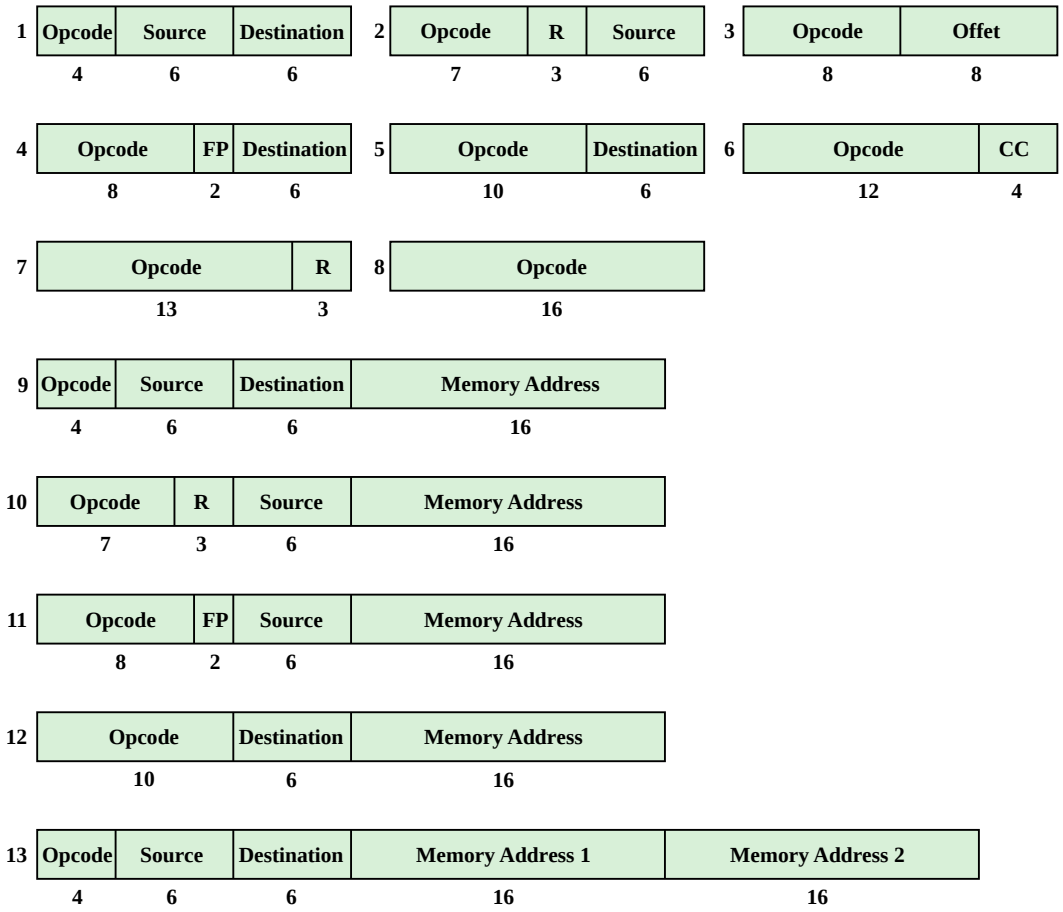


I = indirect bit

Variable-Length Instructions

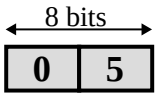
- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
  - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
  - Sometimes multiple instructions are fetched

Instruction Formats for the PDP-11

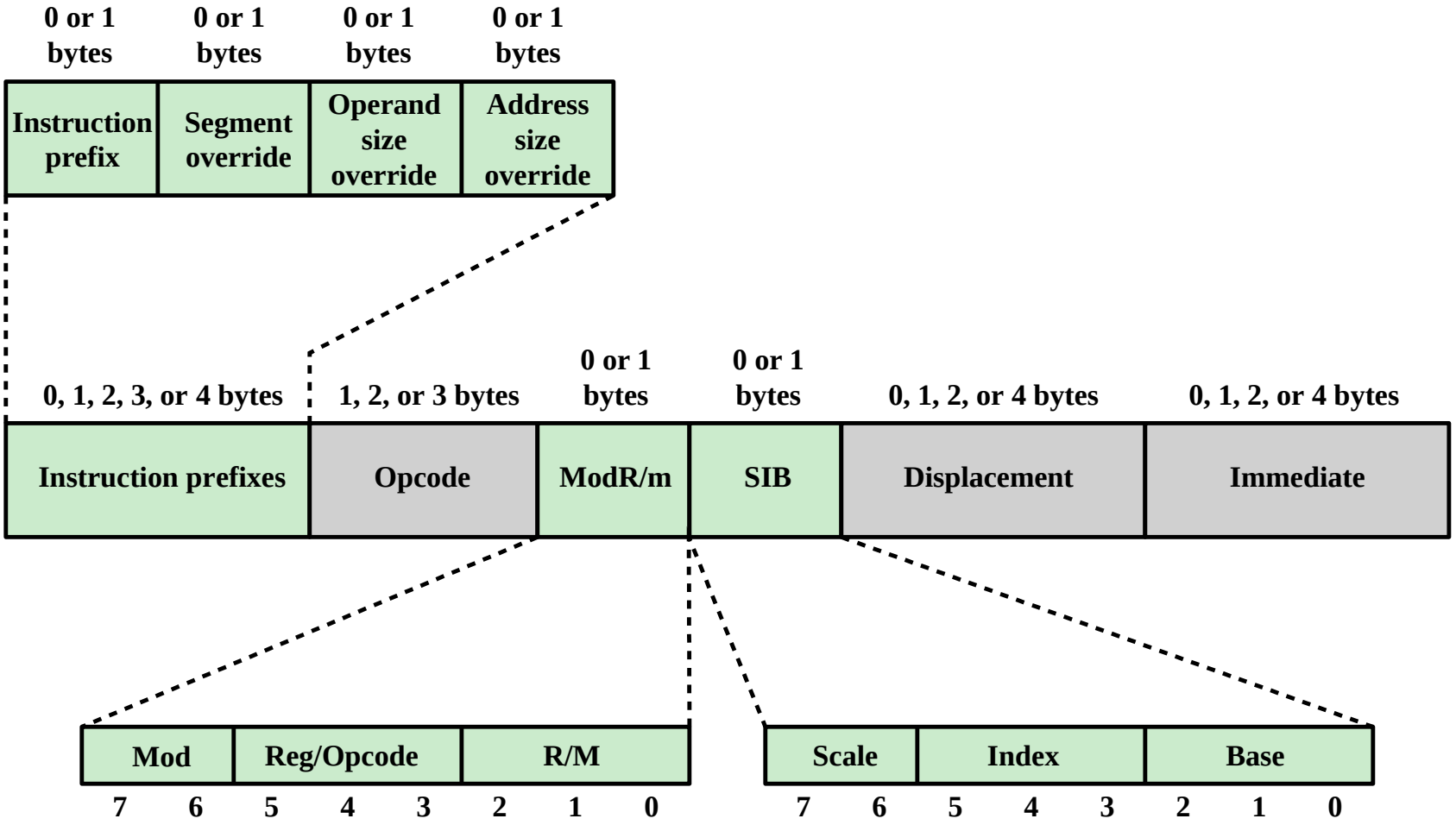


Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

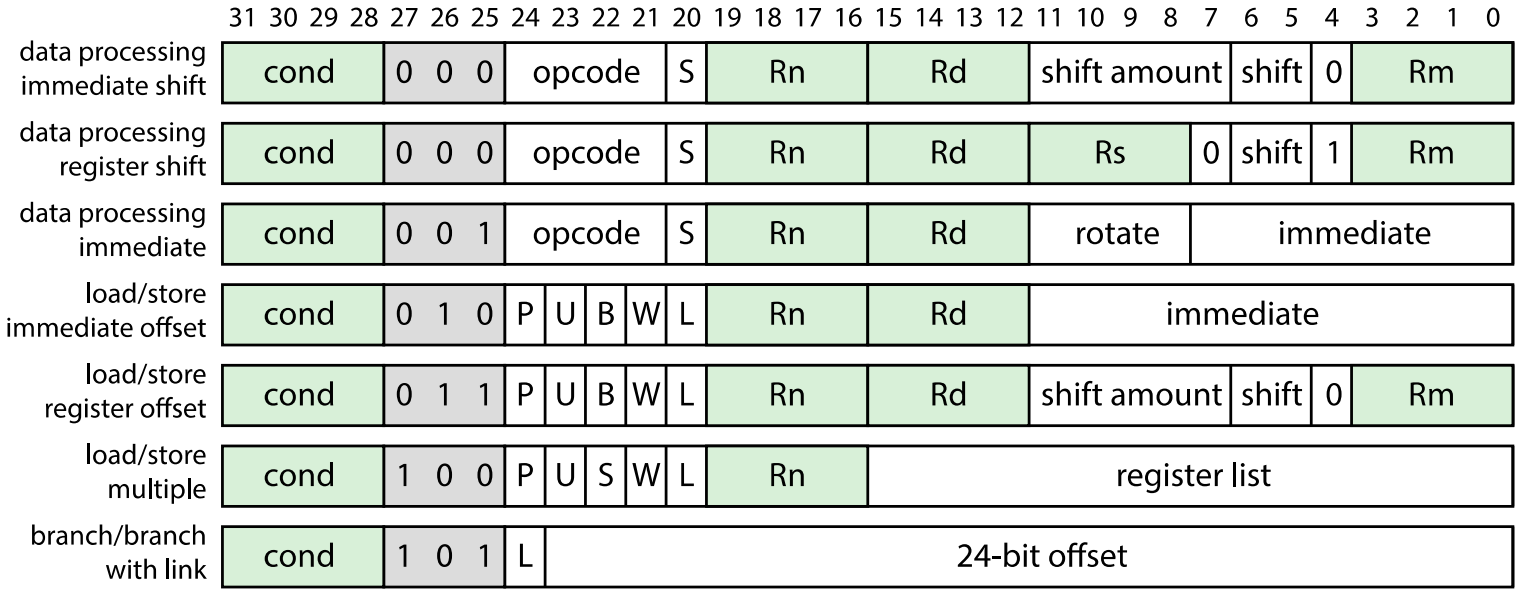
Example of VAX Instructions

Hexadecimal Format	Explanation	Assembler Notation and Description
	Opcode for RSB	RSB Return from subroutine
D 4 5 9	Opcode for CLRL Register R9	CLRL R9 Clear register R9
B 0 C 4 6 4 0 1 A B 1 9	Opcode for MOVW Word displacement mode, Register R4 356 in hexadecimal Byte displacement mode, Register R11 25 in hexadecimal	MOVW 356(R4), 25(R11) Move a word from address that is 356 plus contents of R4 to address that is 25 plus contents of R11
C 1 0 5 5 0 4 2 D F [ ]	Opcode for ADDL3 Short literal 5 Register mode R0 Index prefix R2 Indirect word relative (displacement from PC) Amount of displacement from PC relative to location A	ADDL3 #5, R0, @A[R2] 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, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

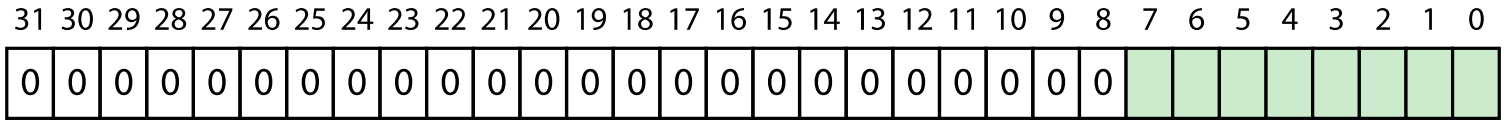
P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

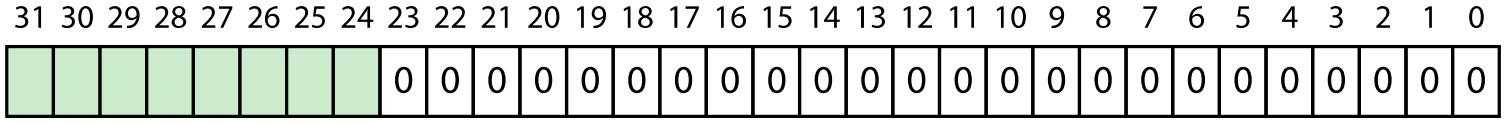
L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

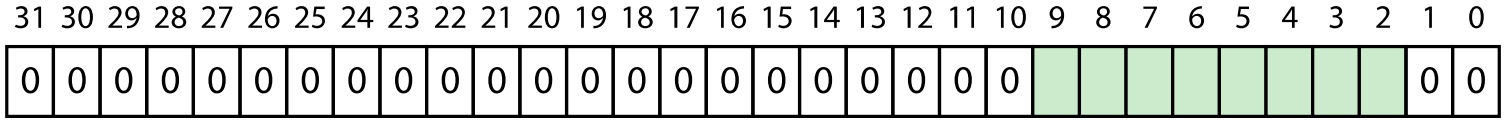
Examples of Use of ARM Immediate Constants



ror #—0 range 0 through 0x000000FF—step 0x00000001

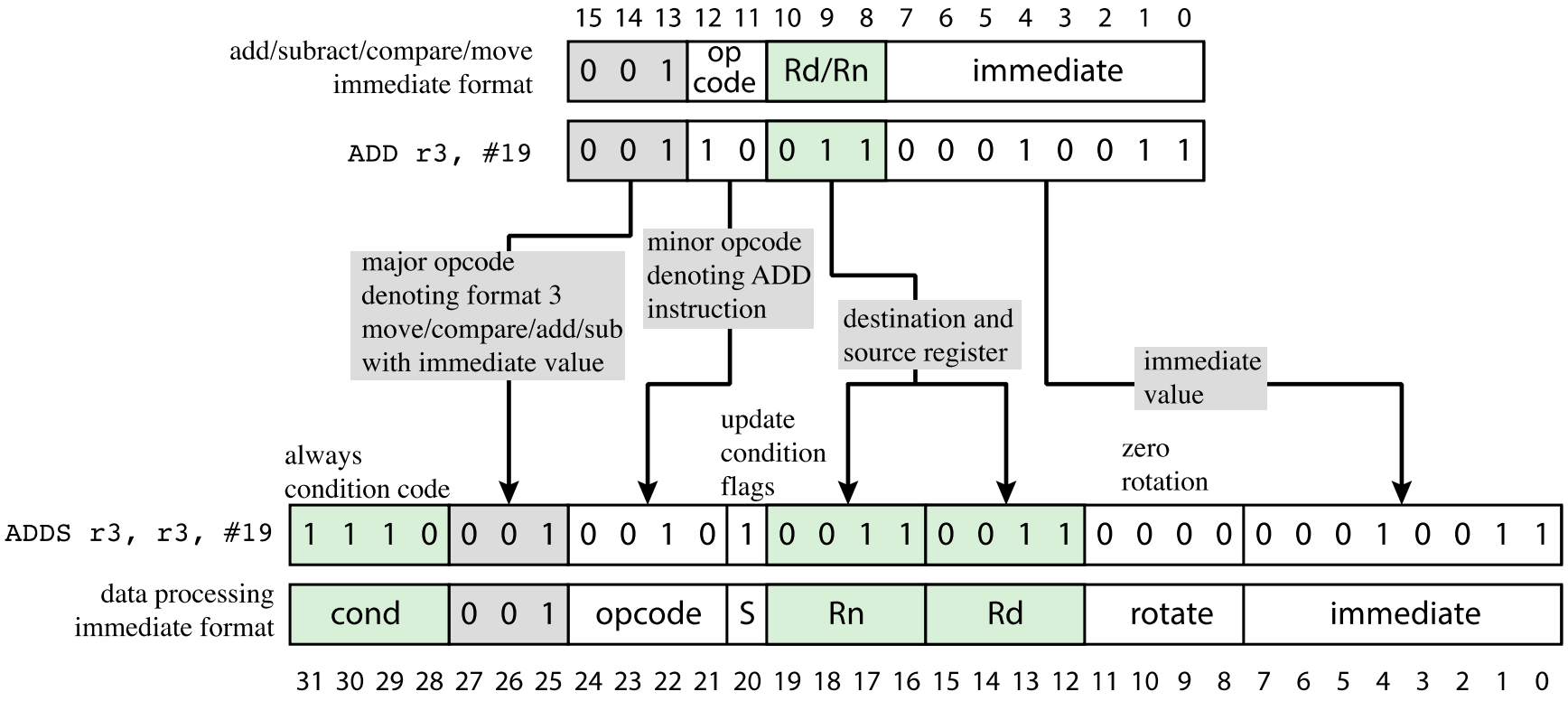


ror #8—range 0 through 0xFF000000—step 0x01000000



ror #30—range 0 through 0x000003FC—step 0x00000004

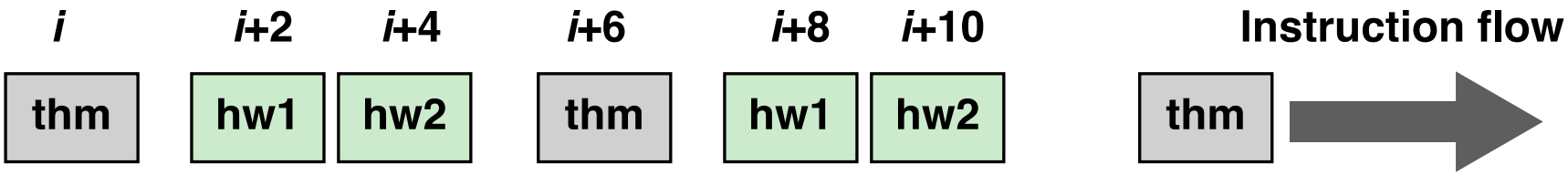
Expanding a Thumb ADD Instruction into its ARM Equivalent



Thumb-2 Instruction Set

- The only instruction set available on the Cortex-M microcontroller products
- Is a major enhancement to the Thumb instruction set architecture (ISA)
  - Introduces 32-bit instructions that can be intermixed freely with the older 16-bit Thumb instructions
  - Most 32-bit Thumb instructions are unconditional, whereas almost all ARM instructions can be conditional
  - Introduces a new If-Then (IT) instruction that delivers much of the functionality of the condition field in ARM instructions
- Delivers overall code density comparable with Thumb, together with the performance levels associated with the ARM ISA
- Before Thumb-2 developers had to choose between Thumb for size and ARM for performance

Thumb-2 Encoding



Halfword 1 [15:13]	Halfword1 [12:11]	Length	Functionality
Not 111	xx	16 bits (1 halfword)	16-bit Thumb instruction
111	00	16 bits (1 halfword)	16-bit Thumb unconditional branch instruction
111	Not 00	32 bits (2 halfwords)	32-bit Thumb-2 instruction