Chapter 11
Instruction Sets:
Addressing Modes and Formats

2020.6
Howon Kim

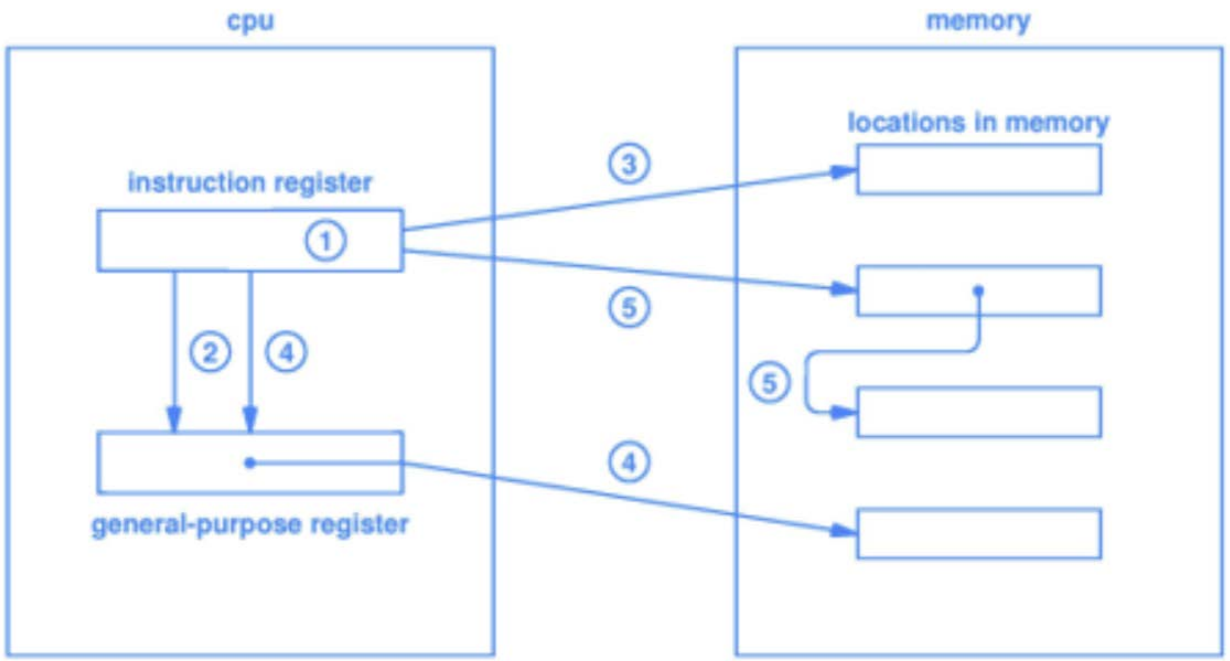
- 정보보호 및 지능형 IoT연구실 - http://infosec.pusan.ac.kr
- 부산대 지능형융합보안대학원 - http://aisec.pusan.ac.kr

Addressing Modes

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

What is Addressing Mode ?

- The CPU can access data(operand) in various ways.
- The data could be in a register, or in memory(RAM or ROM), or be provided as an immediate value.
- The various ways of access data(operand) are called addressing mode.



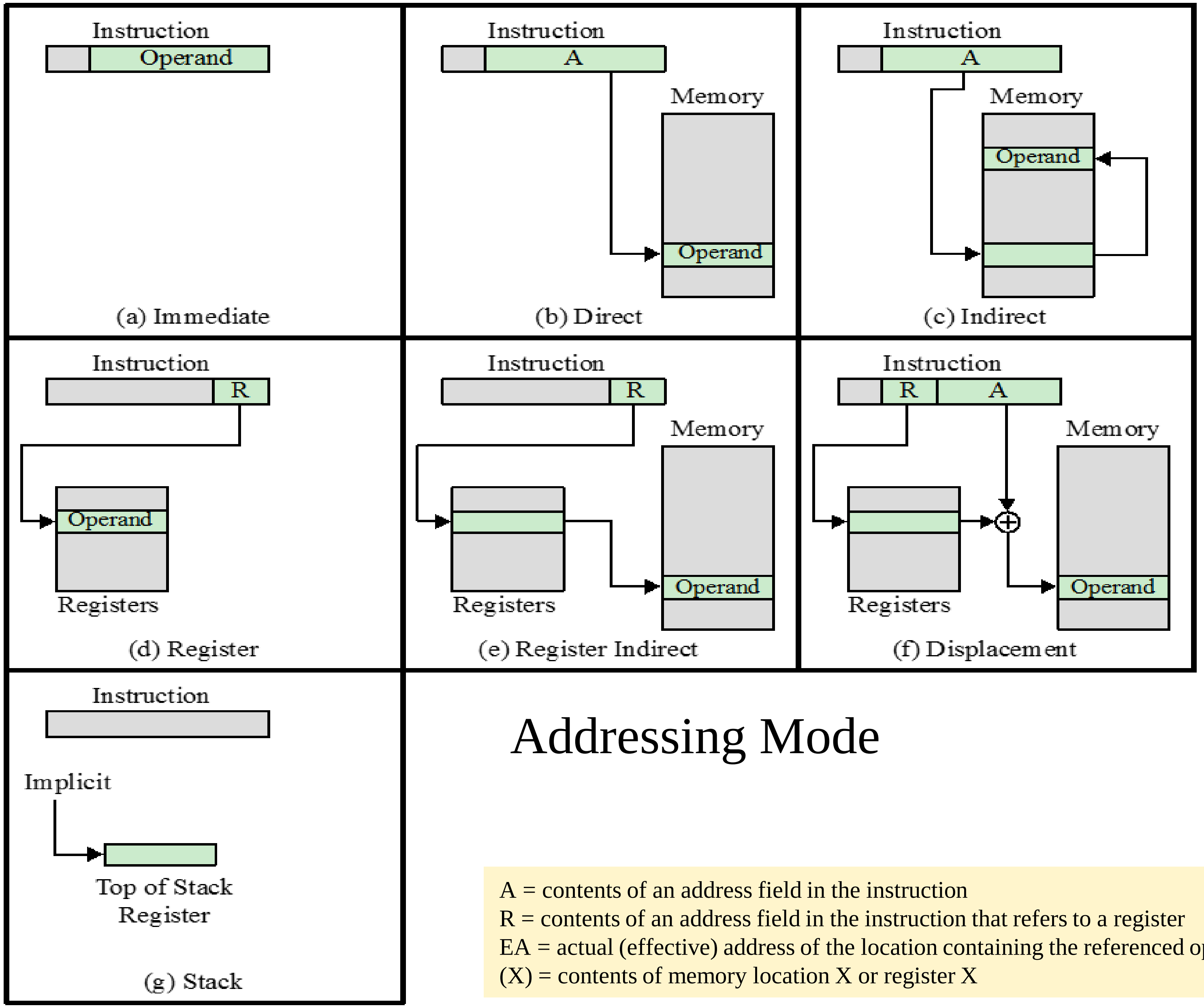
1: immediate mode (명령어에 이미 operand 주소 있음)

2: Register 모드

3: Direct memory 모드

4: Register indirect (Indirect register) 모드

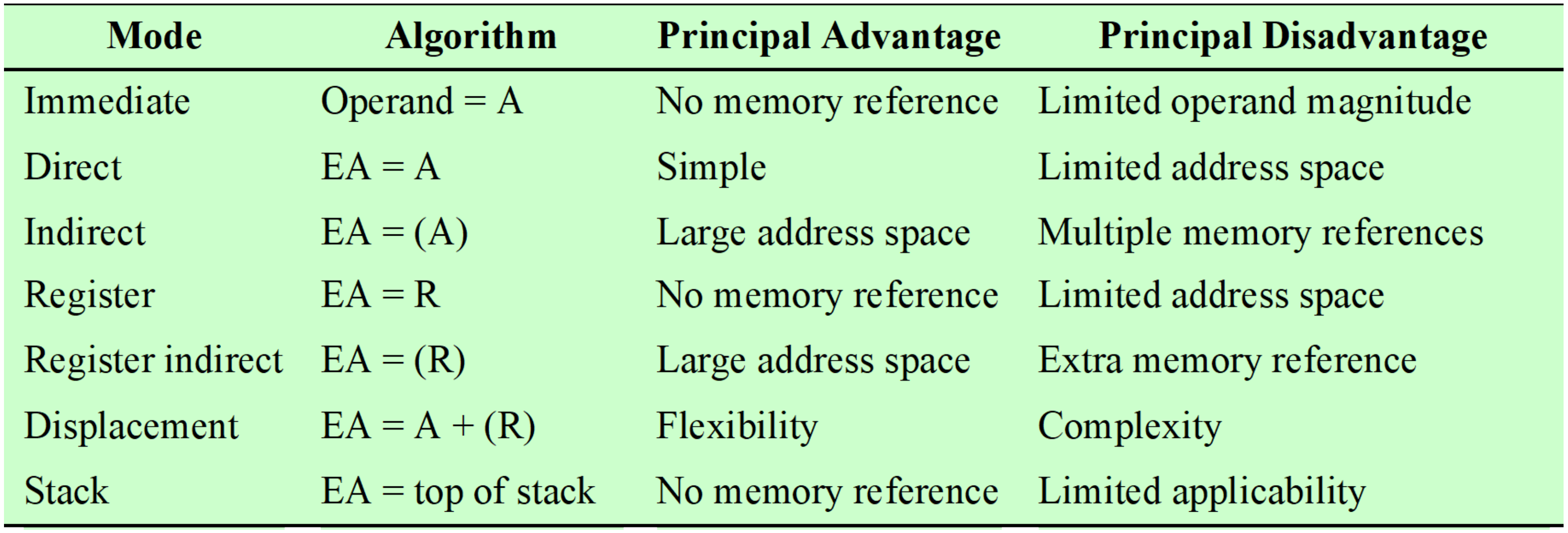
5: Indirect memory 모드



Addressing Mode

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

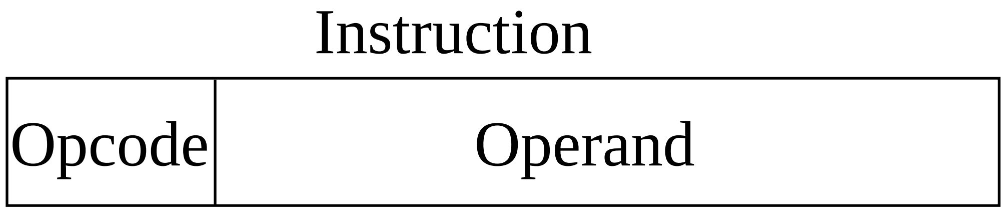
Basic Addressing Modes



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

Immediate Addressing

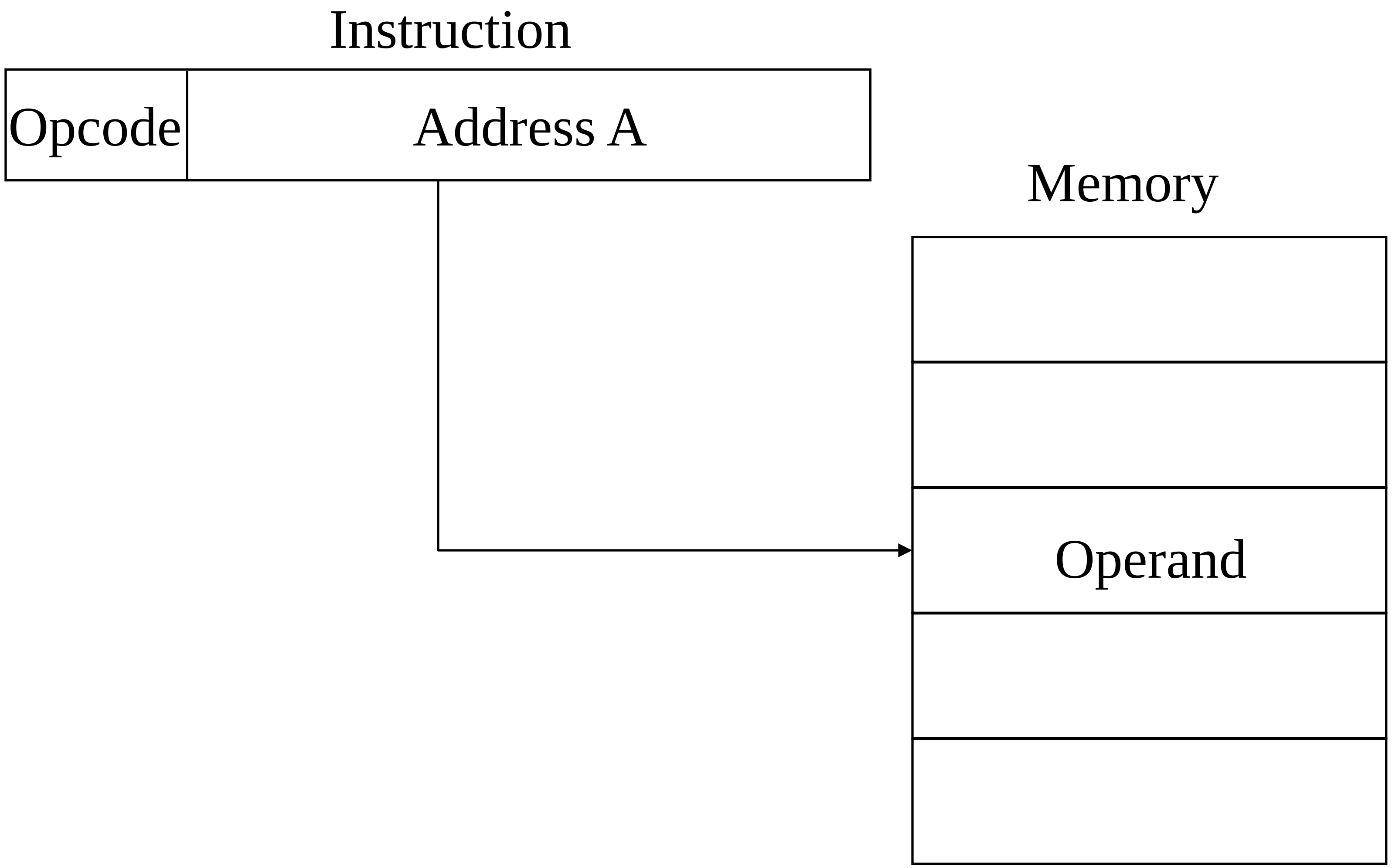
- Operand is part of instruction
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



Direct Addressing

- Address 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)

  ‘( )’ is interpreted as ‘contents of’

  - 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

( ) 괄호의 해석은 "실제 주소는 괄호 안에 있는 값이 가리키는 곳에 있다"라고 풀어서 생각하면 됨.

즉, EA = (A)라고 하면, 실제 원하는 값이 있는 주소는 A가 가리키는 곳에 있음.

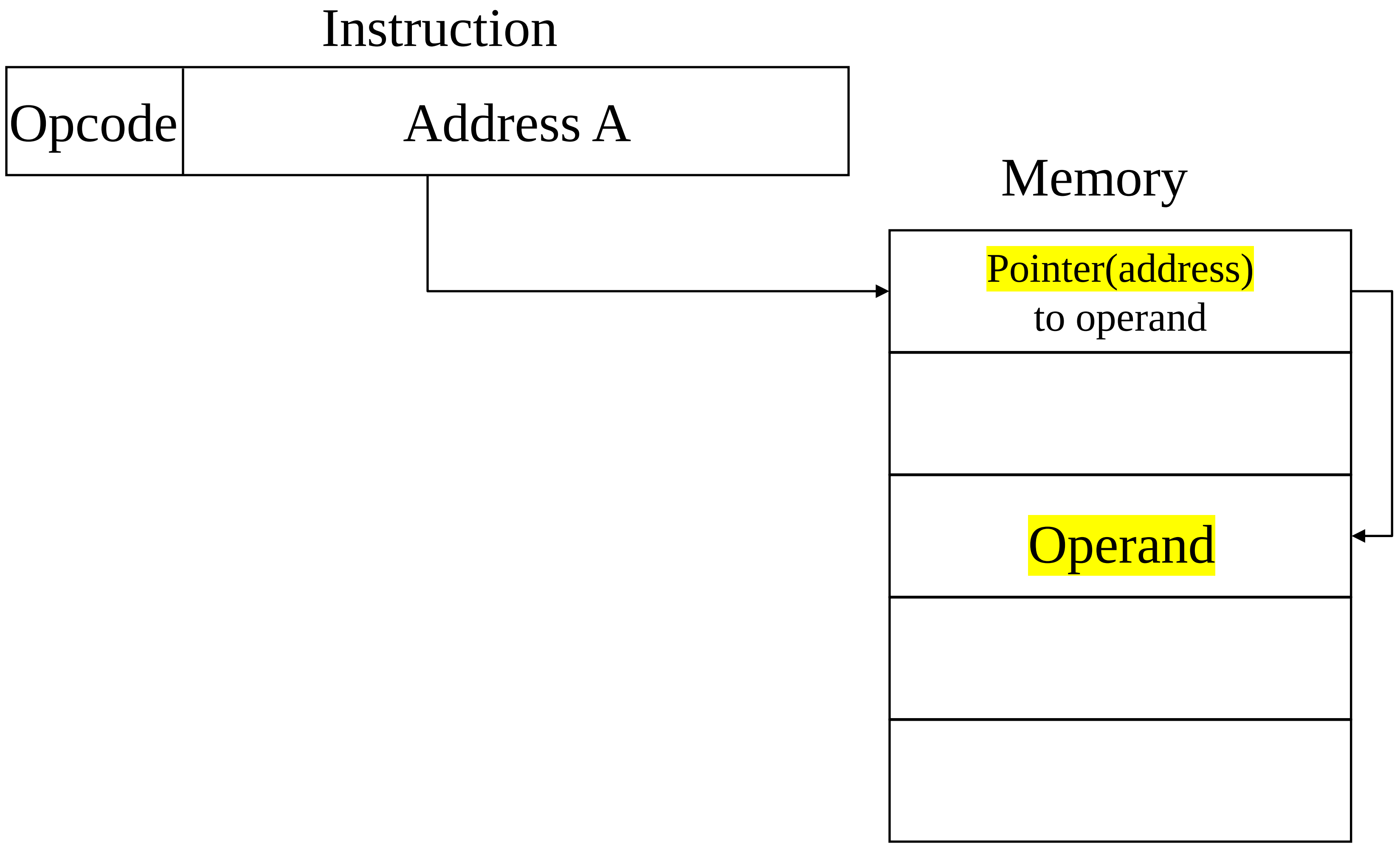
예로서)

•EA = (10)이라고 하면, 주소 0x10 번지에 우리가 원하는 값의 실제 주소가 있음.

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



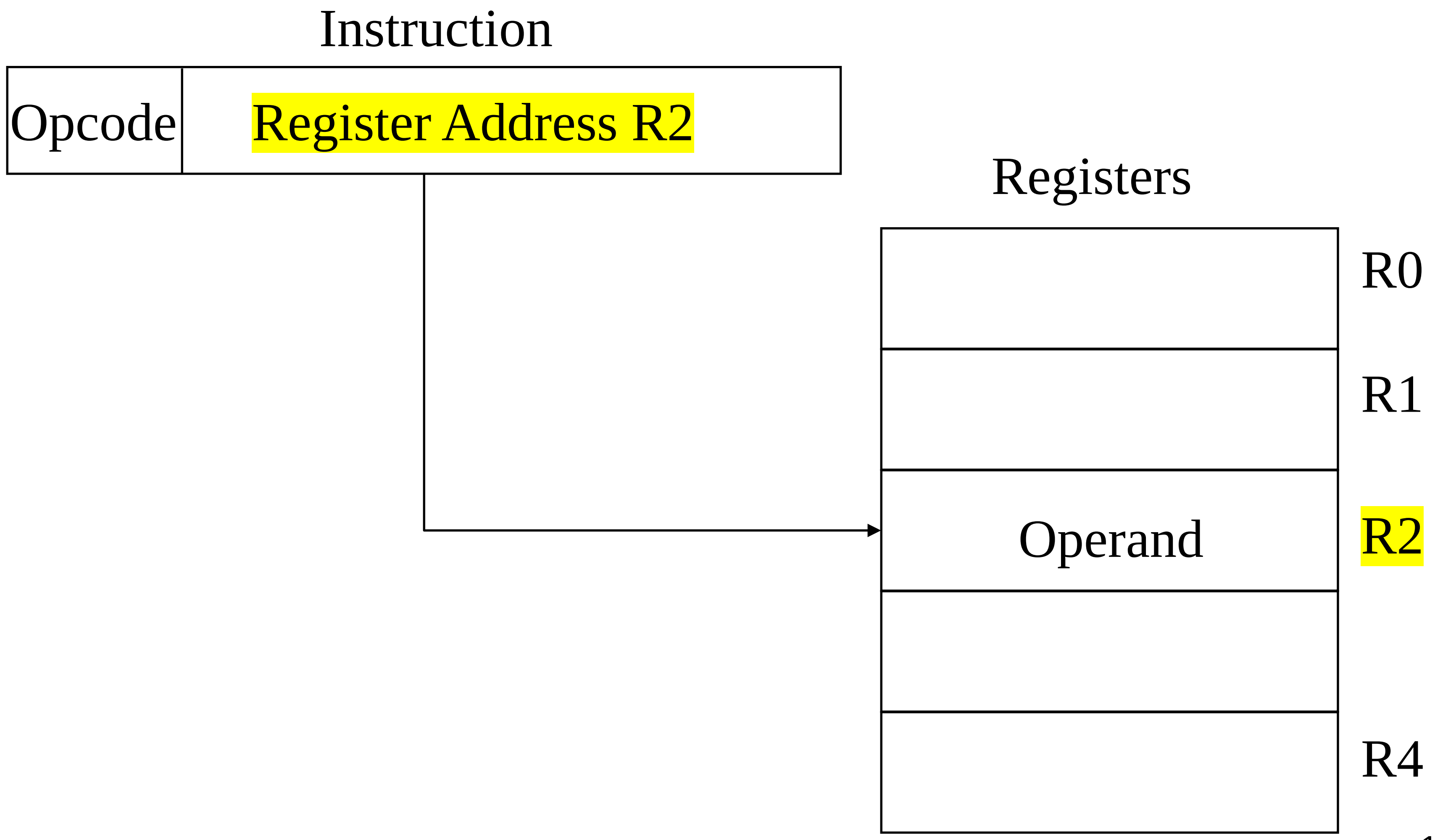
Register Addressing (1)

- Address field in register refers to the effective address of the operand
  - specifically, a register rather than a main memory address
- 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
  - C programming
    - register int a;
- c.f. Direct addressing

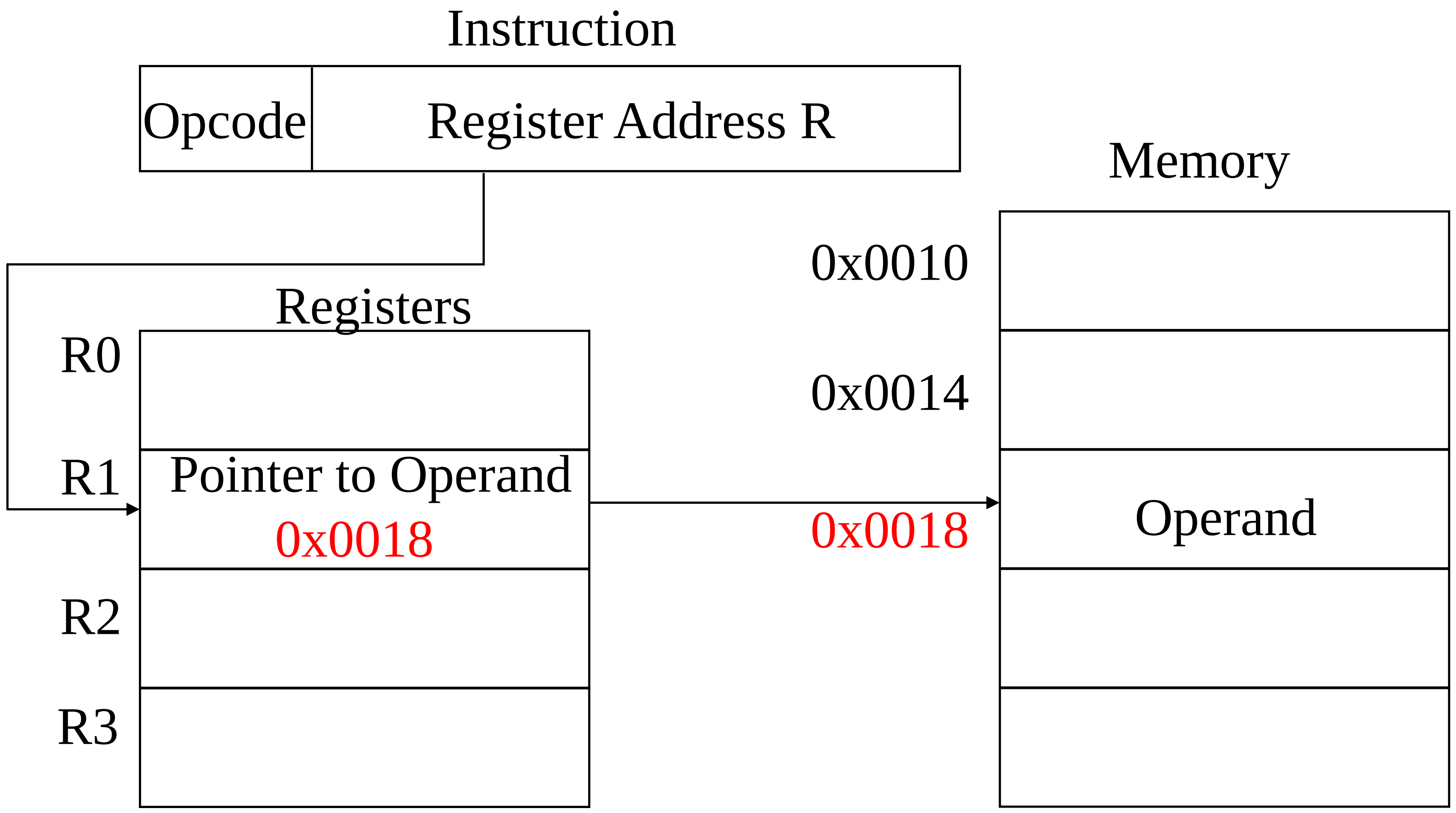
Register Addressing Diagram



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

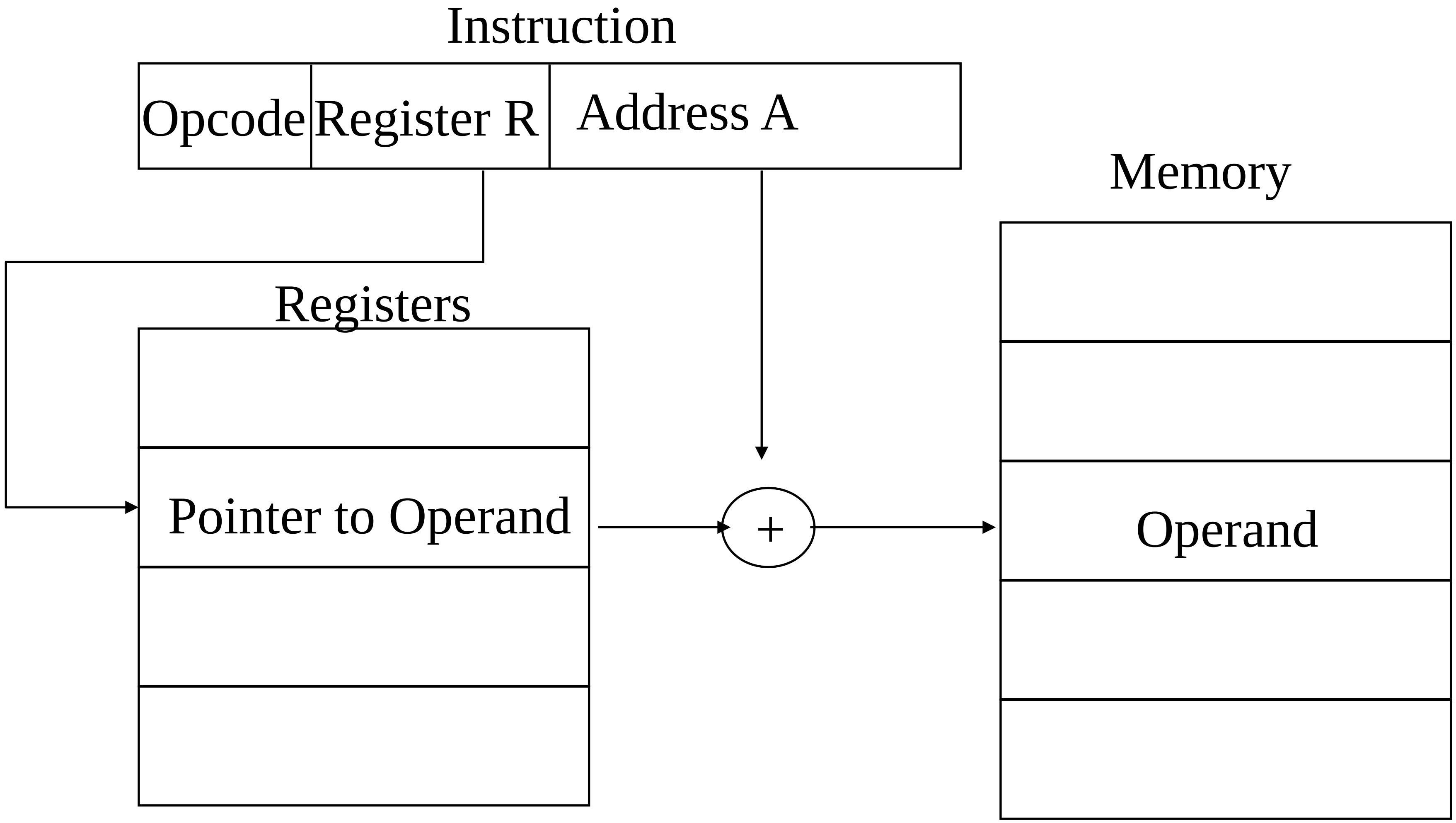


Displacement Addressing

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

- Typical 3 types of displacement addressing
  - Relative addressing
  - Base-register addressing
  - Indexing

Displacement Addressing Diagram



(1) Relative Addressing

- A version of displacement addressing
- The implicitly referenced register is the PC!
  - R = Program counter, PC
- EA = A + (PC)

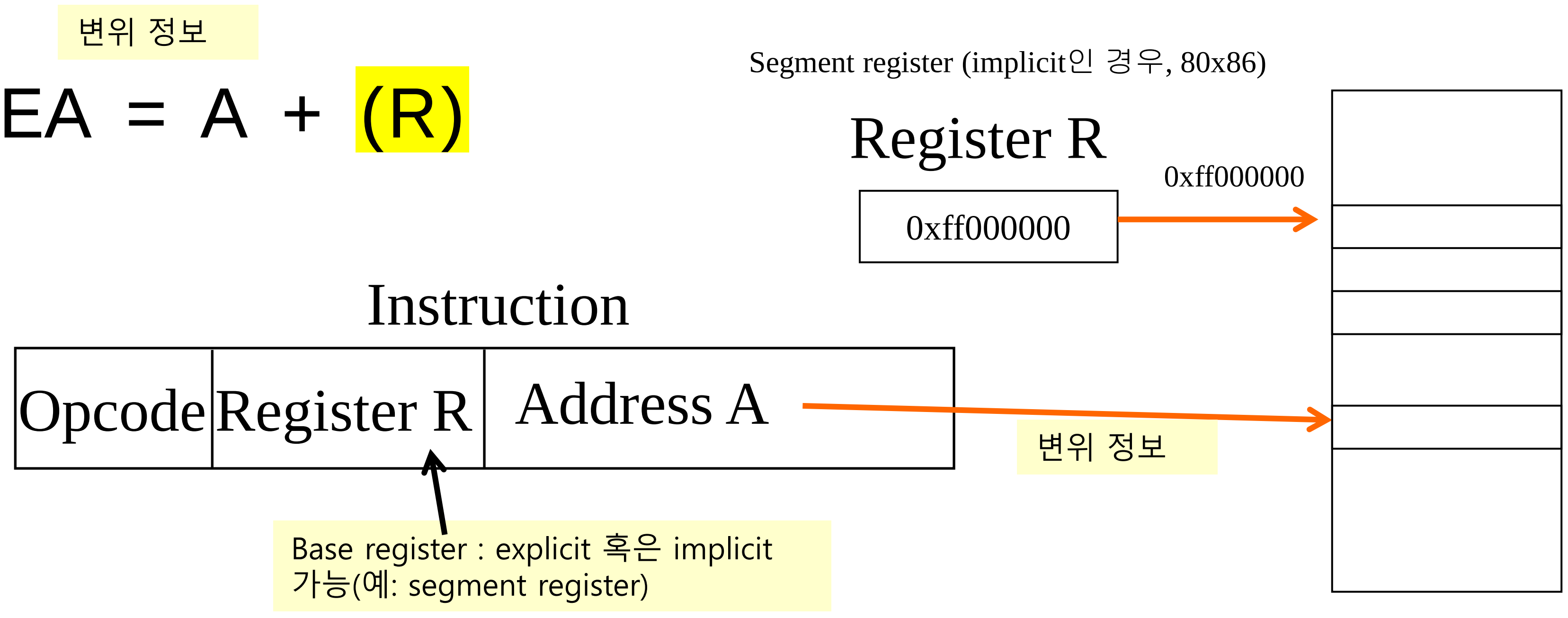
  explicit implicit
  - i.e. the next instruction address (PC) is added to the address field (A) to produce the EA
- Relative addressing exploits the concept of locality
  - If most memory references are relatively near to the instruction being executed(PC!), then the use of relative addressing saves address bits in the instruction.

작은 값을 가지는 A를 instruction에서 제공하면 되므로

(2) Base-Register Addressing

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

변위 정보는 instruction에서 제공(A)



변위 정보
Segment register (implicit인 경우, 80x86)
Register R
0xff000000
0xff000000
Instruction
Opcode
Register R
Address A
변위 정보
Base register : explicit 혹은 implicit 가능(예: segment register)

(3) Indexed Addressing

- A = base
- **R = displacement**

변위 정보는 register에서 제공(R)

  - Opposite to base-register addressing
- EA = A + (R)
- Good for accessing arrays
  - EA = A + (R)
  - (R)++
- Example of Indexing
  - 읽고자 하는 다수의 명령어가 주소 A부터 위치해 있는 경우: A, A+1, A+2, …계산 필요함
  - A는 instruction의 address field에 주소값이 주어지고
  - Index 값은 register에 의해 +1, +2, +3,… 계산되면, 해당 instruction을 불러오는게 용이함
  - 만약, 전용 index register에 의해 1 clk만에 +1, +2, +3… 이 된다면 더 효율적일 것임 → 어떤 프로세서는 **autoindexing** 으로 이를 지원함

Combinations – Autoindexing

- 일반적인 register로 Autoindexing을 하기위해선 다음 동작 필요
  - EA = A + (R)
  - (R) ← (R) + 1
- 하지만, 만약 특정 register(index register)가 indexing 기능을 제공하는 경우, Autoindexing이 가능해 짐
- 또한, 특정 프로세서에서는 "indirect addressing + indexing" 기능도 제공함. Postindexing과 Preindexing 존재

Combinations – Autoindexing

- Postindex :indirect 주소 값 얻은 후, index 값을 계산함
- EA = (A) + (R)

주소필드의 내용은 실제 주소를 가지는 memory의 위치 정보를 제공함. 이를 얻은 후, register값으로 indexing 함.

  — The contents of the address field are used to access a memory location containing a direct address. (A)
  — then, indexed with the register value (R)
  — 이 기법은 동일한 format을 가지는 여러 블록을 처리하는데 유용함. 즉, 각 블록의 주소값만 (A)로 제공하면, 그 이후에는 알아서 처리함

- Preindex : indirection(간접주소지정방식) 이전에 indexing이 이뤄짐
- EA = (A+(R))
  — First, calculate the address of the operand, that is the index
    – A+ (R)
  — Ex) multiway branch table에 유용함 : branching to one of a number of locations depending on condition
    – A table of addresses can be set up starting at location ‘A’. By indexing into this table, the required location can be found.

Stack Addressing

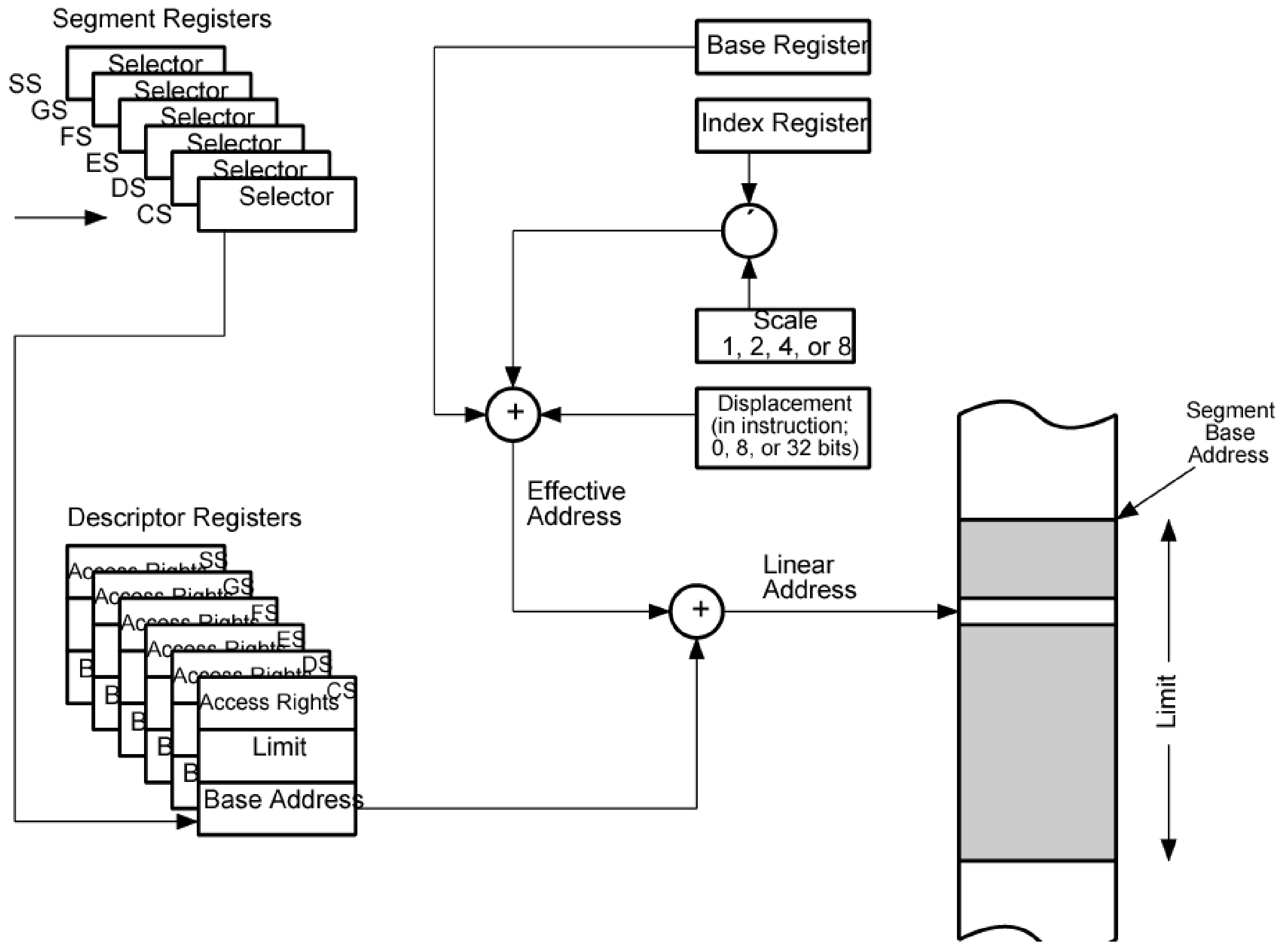
- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add

- 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

Pentium 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

Pentium Addressing Mode Calculation



One can use either flat memory model or segmented memory mode. With the flat memory model, memory appears to a program as a single, continuous address space, called a linear address space. Code (a program's instructions), data, and the procedure stack are all contained in this address space. The linear address space is byte addressable, with addresses running contiguously from 0 to 2^{32-1}.With the segmented memory mode, memory appears to a program as a group of independent address spaces called segments. When using this model, code, data, and stacks are typically contained in separate segments. To address a byte in a segment, a program must issue a logical address, which consists of a segment selector and an offset

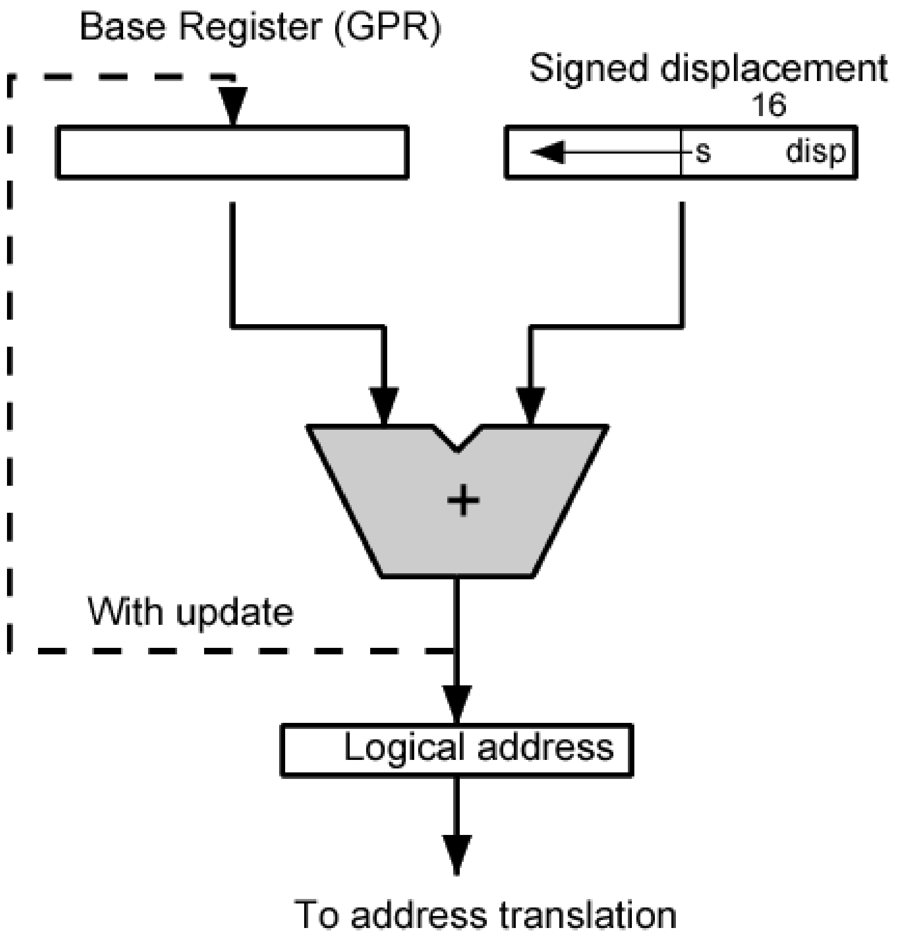
http://flint.cs.yale.edu/cs422/doc/pc-arch.html

PowerPC Addressing Modes

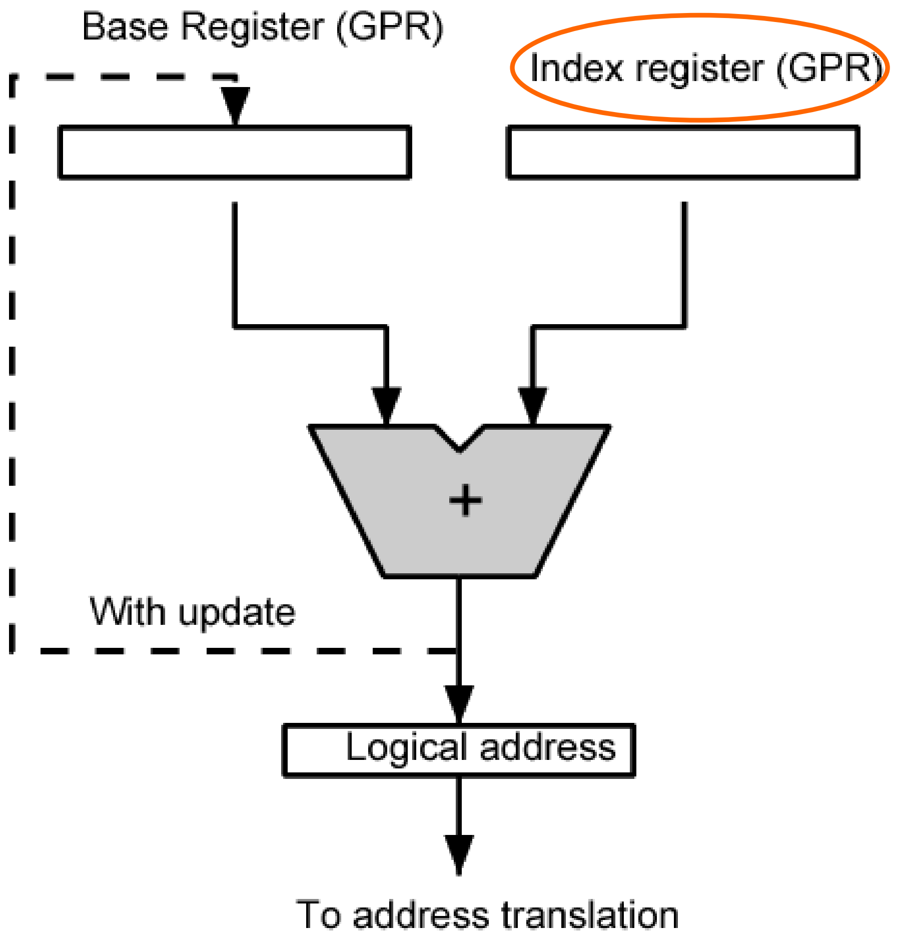
- Load/store architecture
  - Indirect
    - Instruction includes 16 bit displacement to be added to base register (may be GP register)
    - Can replace base register content with new address

      Useful for progressive indexing of arrays in loops
  - Indirect indexed
    - Instruction references base register and index register (both may be GP)
    - EA is sum of contents
- Branch address
  - Absolute
  - Relative
  - Indirect
- Arithmetic
  - Operands in registers or part of instruction
  - Floating point is register only

PowerPC Memory Operand Addressing Modes



(a) Indirect Adressing



(b) Indirect Indexed Addressing

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

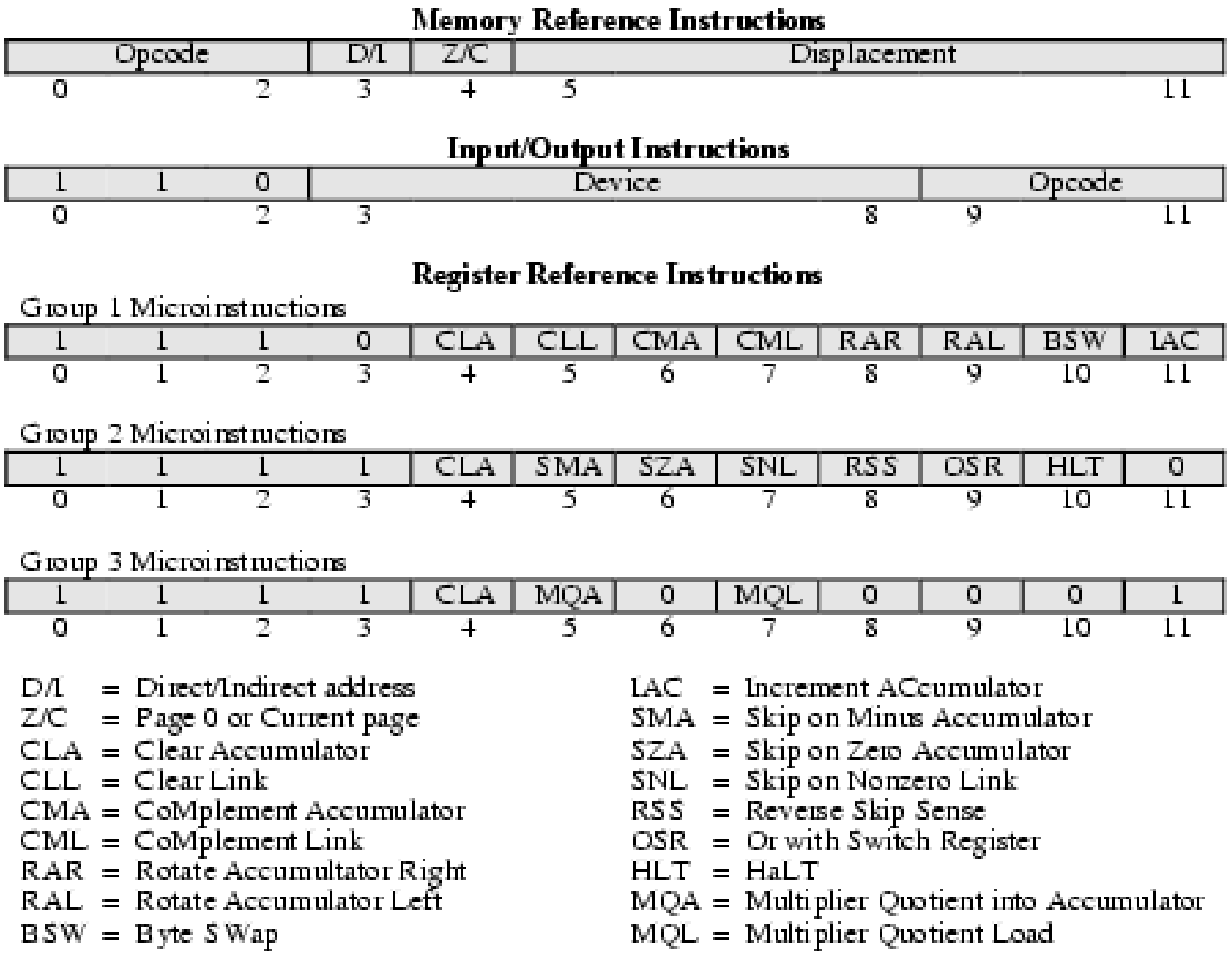
- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire and saving space

Allocation of Bits

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

Q&A

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

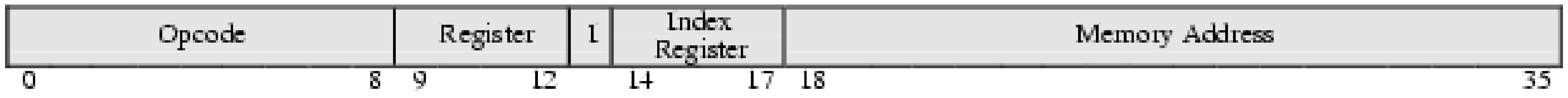
1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

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

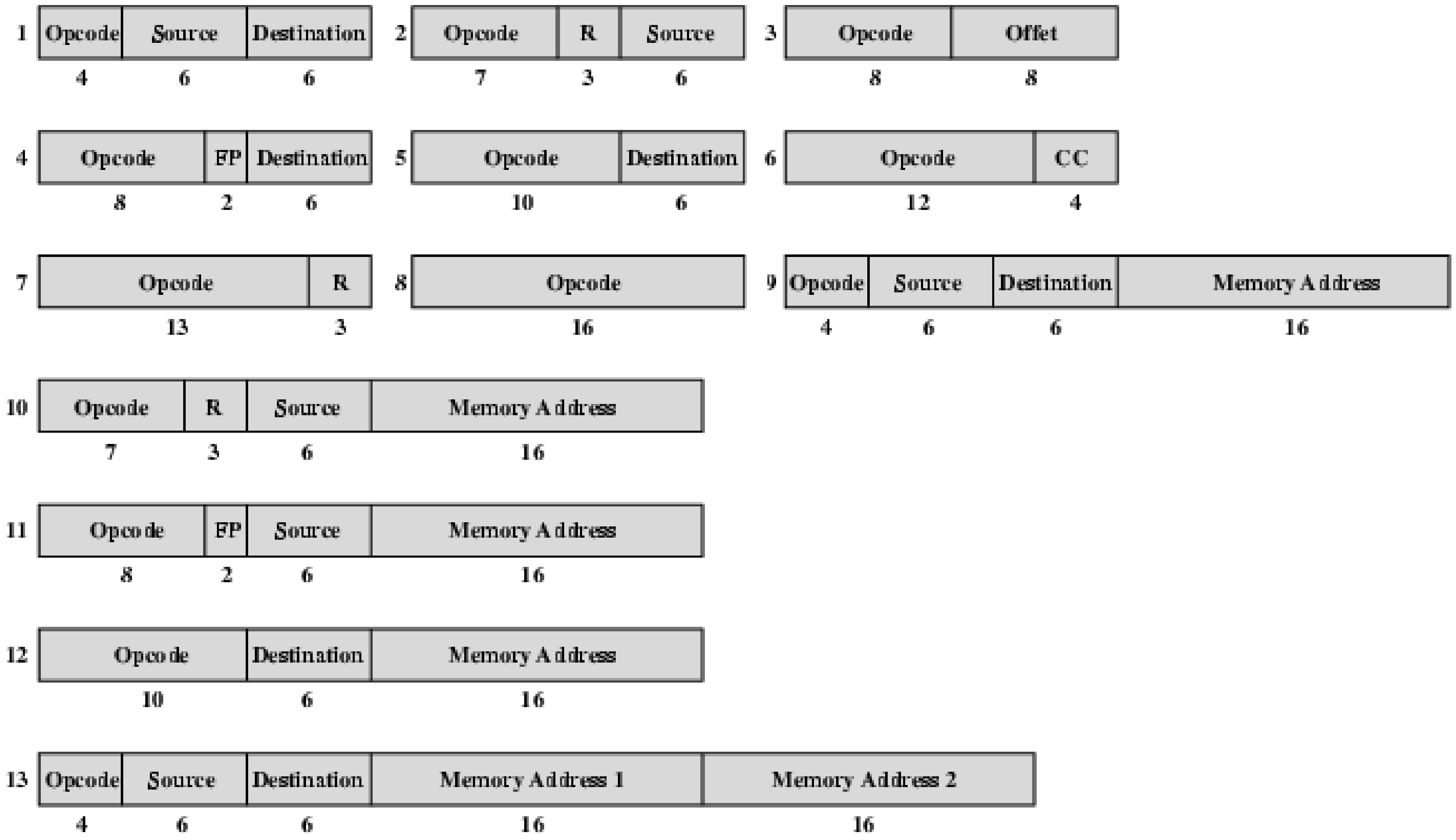
D/I	=	Direct/Indirect address	IAC	=	Increment ACcumulator
Z/C	=	Page 0 or Current page	SMA	=	Skip on Minus Accumulator
CLA	=	Clear Accumulator	SZA	=	Skip on Zero Accumulator
CLL	=	Clear Link	SNL	=	Skip on Nonzero Link
CMA	=	CoMplement Accumulator	RSS	=	Reverse Skip Sense
CML	=	CoMplement Link	OSR	=	Or with Switch Register
RAR	=	Rotate Accumultator Right	HLT	=	HaLT
RAL	=	Rotate Accumulator Left	MQA	=	Multiplier Quotient into Accumulator
BSW	=	Byte SWap	MQL	=	Multiplier Quotient Load

PDP-10 Instruction Format



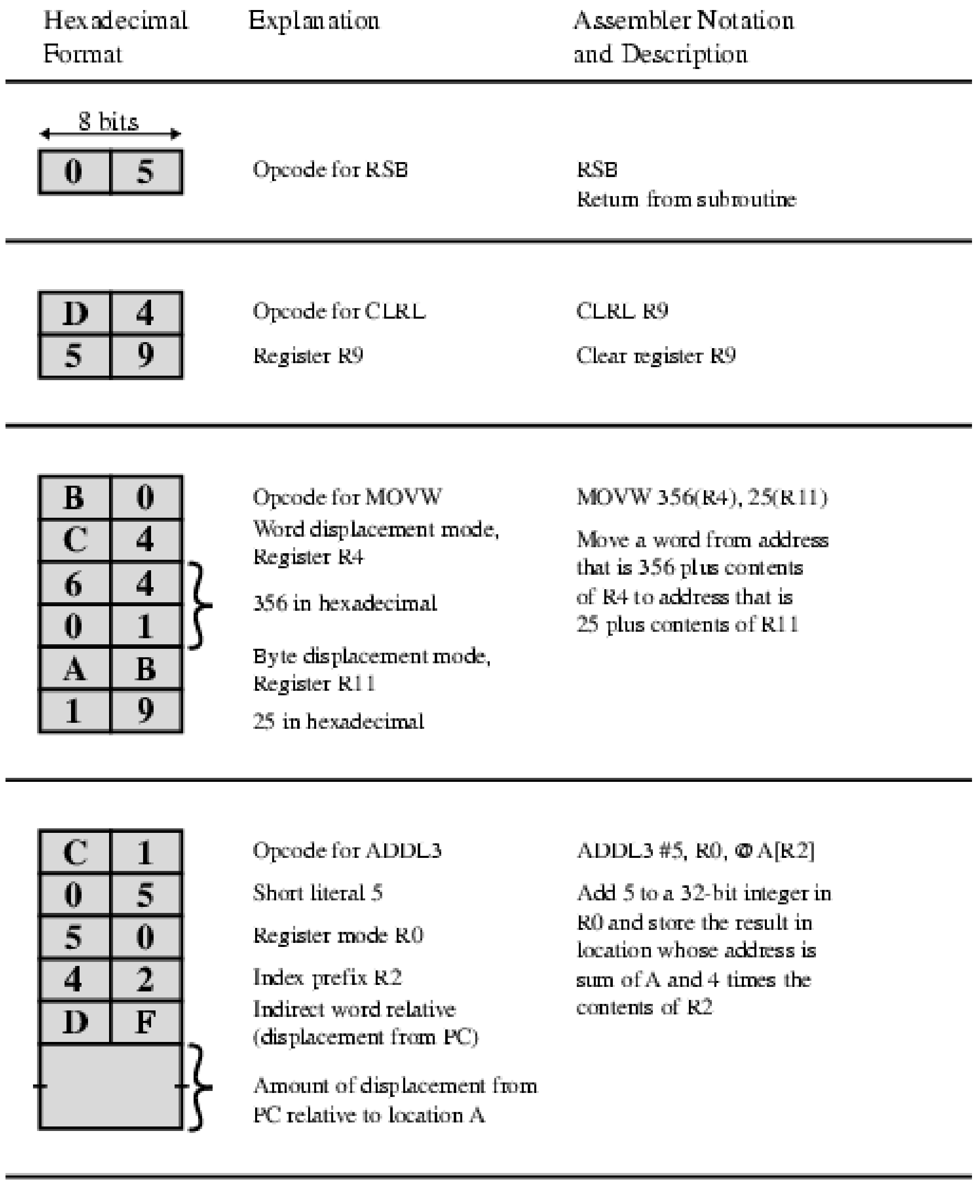
I = indirect bit

PDP-11 Instruction Format


1 Opcode 4 Source 6 Destination 6
2 Opcode 7 R 3 Source 6
3 Opcode 8 Offet 8
4 Opcode 8 FP 2 Destination 6
5 Opcode 10 Destination 6
6 Opcode 12 CC 4
7 Opcode 13 R 3
8 Opcode 16
9 Opcode 4 Source 6 Destination 6 Memory Address 16
10 Opcode 7 R 3 Source 6 Memory Address 16
11 Opcode 8 FP 2 Source 6 Memory Address 16
12 Opcode 10 Destination 6 Memory Address 16
13 Opcode 4 Source 6 Destination 6 Memory Address 1 16 Memory Address 2 16

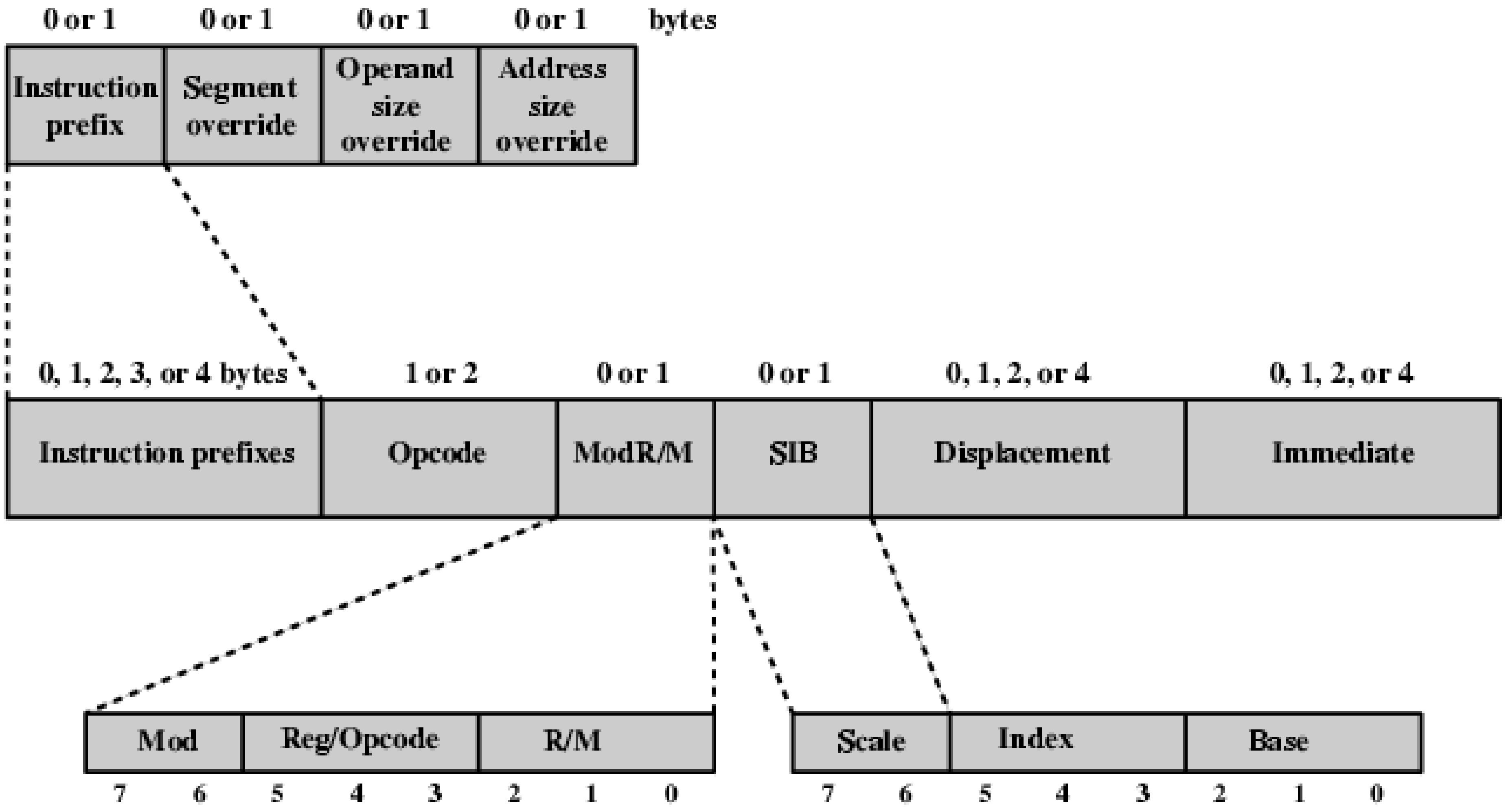
Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

VAX Instruction Examples

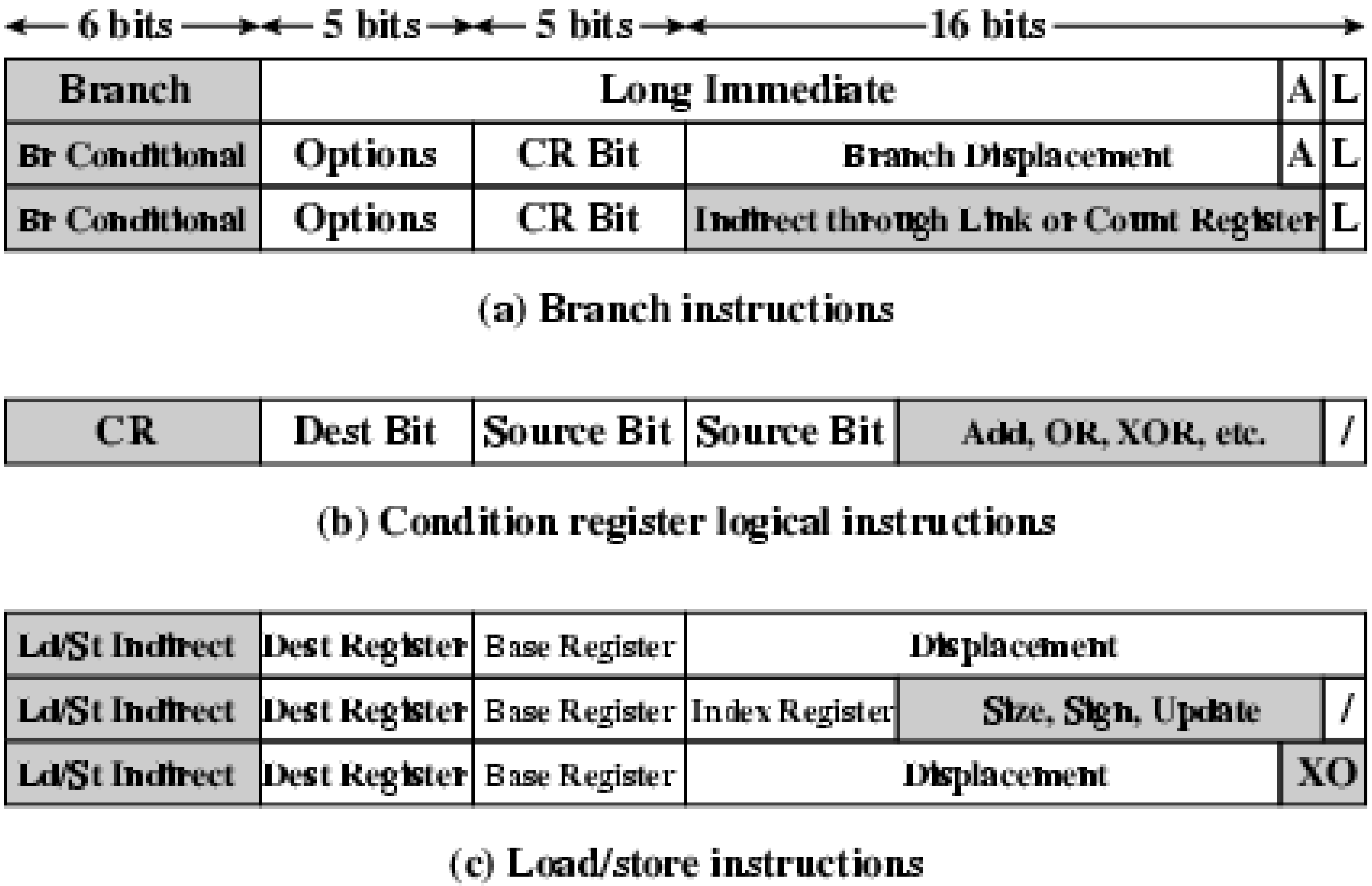


Hexadecimal Format	Explanation	Assembler Notation and Description
8 bits: 0 5	Opcode for RSB	RSB Return from subroutine
D 4	Opcode for CLRL	CLRL R9
5 9	Register R9	Clear register R9
B 0	Opcode for MOVW	MOVW 356(R4), 25(R11)
C 4	Word displacement mode, Register R4	Move a word from address that is 356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

Pentium Instruction Format



PowerPC Instruction Formats (1)



6 bits	5 bits	5 bits	16 bits		
Branch	Long Immediate			A	L
Br Conditional	Options	CR Bit	Branch Displacement	A	L
Br Conditional	Options	CR Bit	Indirect through Link or Count Register		L

(a) Branch instructions

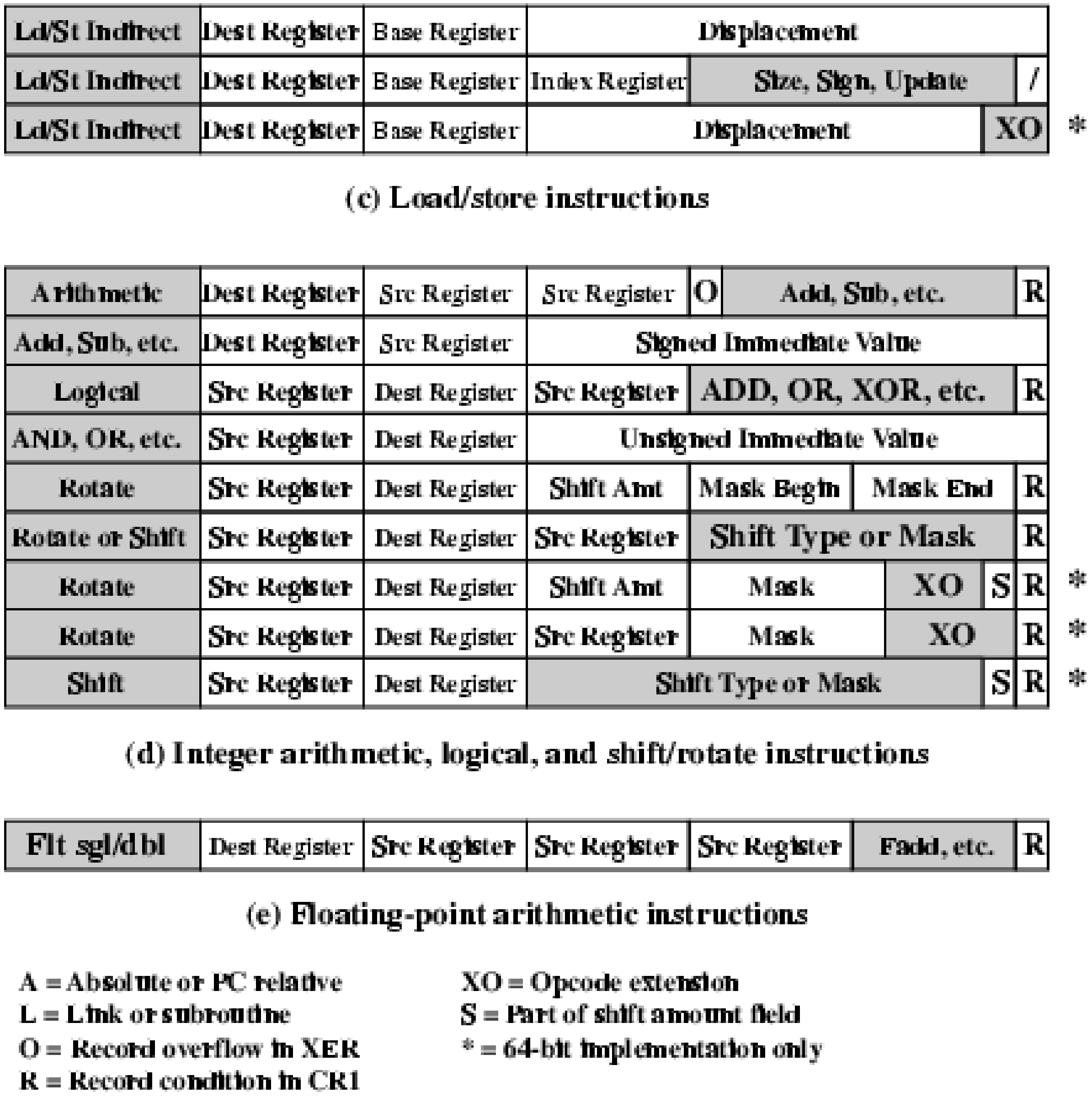
CR	Dest Bit	Source Bit	Source Bit	Add, OR, XOR, etc.	/

(b) Condition register logical instructions

Ld/St Indirect	Dest Register	Base Register	Displacement		
Ld/St Indirect	Dest Register	Base Register	Index Register	Size, Sign, Update	/
Ld/St Indirect	Dest Register	Base Register	Displacement		XO

(c) Load/store instructions

PowerPC Instruction Formats (2)



Ld/St Indirect	Dest Register	Base Register	Displacement			
Ld/St Indirect	Dest Register	Base Register	Index Register	Size, Sign, Update	/	
Ld/St Indirect	Dest Register	Base Register	Displacement		XO	*

(c) Load/store instructions

Arithmetic	Dest Register	Src Register	Src Register	O	Add, Sub, etc.		R	
Add, Sub, etc.	Dest Register	Src Register	Signed Immediate Value					
Logical	Src Register	Dest Register	Src Register	ADD, OR, XOR, etc.			R	
AND, OR, etc.	Src Register	Dest Register	Unsigned Immediate Value					
Rotate	Src Register	Dest Register	Shift Amt	Mask Begin	Mask End		R	
Rotate or Shift	Src Register	Dest Register	Src Register	Shift Type or Mask			R	
Rotate	Src Register	Dest Register	Shift Amt	Mask	XO	S	R	*
Rotate	Src Register	Dest Register	Src Register	Mask	XO		R	*
Shift	Src Register	Dest Register	Shift Type or Mask			S	R	*

(d) Integer arithmetic, logical, and shift/rotate instructions

Flt sgl/dbl	Dest Register	Src Register	Src Register	Src Register	Fadd, etc.	R

(e) Floating-point arithmetic instructions

A = Absolute or PC relative	XO = Opcode extension
L = Link or subroutine	S = Part of shift amount field
O = Record overflow in XER	* = 64-bit implementation only
R = Record condition in CR1	