Instruction Sets:
– Characteristics and Functions
– Addressing Modes and Formats

Chapters 13 & 14

Based on:
William Stallings
Computer Organization and Architecture, 11th Global Edition

Patterson and Hennessy
Computer Organization and Design
5th Edition

Instruction Set

- The vocabulary of commands understood by a given architecture

- Different computers have different instruction sets but with many aspects in common
  - all computers are constructed from hardware technologies based on similar underlying principles
  - there are a few basic operations that all computers must provide
  - common goal: find a language that makes it easy to build the hardware and the compiler while maximizing performance and minimizing cost and energy

- Many modern computers also have simple instruction sets

Elements of a Machine Instruction

- Operation code:
  - specifies operation to be performed (e.g., ADD, I/O)
  - binary code, known as the operation code, or opcode
- Source operand reference:
  - operation may involve one or more source operands, that is, operands that are inputs for the operation
- Result operand reference:
  - operation may produce a result
- Next instruction reference:
  - tells the processor where to fetch the next instruction after the execution of this instruction is complete

Source and Result Operands

- **Main or virtual memory**
  - main or virtual memory address must be supplied
- **Processor register**
  - processor contains one or more registers that may be referenced by machine instructions
  - if only one register exists, reference to it may be implicit
  - if more than one register exists, then each register is assigned a unique name or number, and the instruction must contain the number of the desired register
- **Immediate**
  - value of the operand is contained in a field in the instruction being executed
- **I/O device**
  - instruction must specify the I/O module and device for the operation

Number of Addresses

- What is the maximum number of addresses one might need in an instruction?

- Arithmetic and logic instructions will require the most operands
  - All arithmetic and logic operations are either unary (one source operand) or binary (two source operands)
  - We would need a maximum of two addresses to reference source operands.
  - The result of an operation must be stored, suggesting a third address, which defines a destination operand.
  - Finally, after completion of an instruction, the next instruction must be fetched, and its address is needed.

- In most architectures, many instructions have one, two, or three operand addresses, with the address of the next instruction being implicit (obtained from the program counter).

Programs to Execute

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

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

Zero-Address Instructions

- Possible to make do with zero addresses for some instructions

- Zero-address instructions are applicable to a special memory organization called a stack
  - A stack is a last-in-first-out set of locations
  - At least the top two elements are in processor registers
  - Zero-address instructions would reference the top two stack elements

- Need to represent expression in postfix notation

- Examples

Utilization of Instruction Addresses

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	$A \leftarrow B$ OP C
2	OP A, B	$A \leftarrow A$ OP B
1	OP A	$AC \leftarrow AC$ OP A
0	OP	$T \leftarrow (T - 1)$ OP T

AC = accumulator
T = top of stack
(T − 1) = second element of stack
A, B, C = memory or register locations

Design Trade-offs

- **Fewer addresses per instruction**
  - instructions that are more primitive, requiring a less complex processor
  - instructions of shorter length
  - programs contain more total instructions, which in general results in longer execution times and longer, more complex programs
- **One-address vs multiple address**
  - with one-address instructions, programmer generally has available only one general-purpose register, the AC
  - with multiple-address instructions, it is common to have multiple general-purpose registers
  - this allows some operations to be performed solely on registers
  - because register references are faster than memory references, this speeds up execution
  - for flexibility and the ability to use multiple registers, most contemporary machines employ a mixture of two- and three- address instructions
- **Other factors complicate this design trade-off**
  - whether an address references a memory location or a register
  - because there are fewer registers, fewer bits are needed for a register reference
  - a machine may offer variety of addressing modes, and the specification of mode takes one or more bits
    - most processor designs involve a variety of instruction formats

Instruction Set Design

- Programmer's means of controlling the processor
- **Operation repertoire**
  - How many and which operations to provide, and how complex operations should be
- **Data types**
  - The various types of data upon which operations are performed
- **Instruction format**
  - Instruction length (in bits), number of addresses, size of various fields, and so on
- **Registers**
  - Number of processor registers that can be referenced by instructions, and their use
- **Addressing**
  - The mode or modes by which the address of an operand is specified

Types of Operands

- Machine instructions operate on data

- The most important general categories of data are
  - Addresses
  - Numbers
  - Characters
  - Logical data

Types of Operations

- Number of different opcodes varies widely from machine to machine
- Same general types of operations found on all machines
- Useful and typical categorization:
  - Data transfer
  - Arithmetic
  - Logical
  - Conversion
  - I/O
  - System control
  - Transfer of control

Type	Operation Name	Description
Data transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT	(complement) Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Type	Operation Name	Description
Input/output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Transfer of control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued

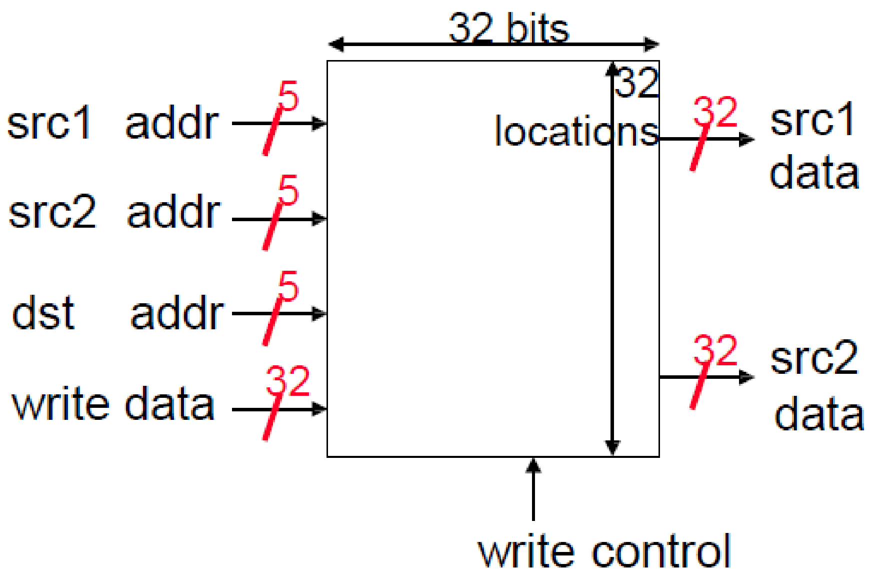
Case Study: MIPS

- Word is 32 bits
- Memory is byte addressable
- Instructions are 32 bits
- Opcode is 6 bits

- 3 Instruction types
  - R-format
  - I-format
  - J-format

Register Operands

- Operands of arithmetic instructions are restricted
  - Must be from a limited number of **registers**
- Arithmetic instructions use register operands
- MIPS has a 32 ×32-bit register file
- Use for frequently accessed data
- Numbered 0 to 31
- Two read ports
- One write port



Memory Operands

- Main memory used for composite data
  - Arrays, structures, …
-
- In MIPS instructions, arithmetic operations occur only on registers
  - MIPS must include instructions that transfer data between memory and registers
  - **data transfer instructions**
- To apply arithmetic operations
  - **Load** values from memory into registers
  - **Store** result from register to memory
- **Load/store architecture**

Data Transfer Instructions

- To access a word in memory, the instruction must supply the memory **address**
- Format of the load instruction:
  - name of the operation
  - followed by the register to be loaded
  - then a constant and register used to access memory
  - The sum of the constant portion of the instruction and the contents of the second register forms the memory address.
- MIPS name for this instruction is **lw**, standing for **load word**
- Example: `lw $t0, 8($s3)`

  `sw $t0, 8($s3)`

Memory

- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4
- MIPS is **Big Endian**

- Example
  - 32-bit hex: 12345678 stored at byte location 184
    - 4 bytes
    - least significant byte containing the value 78
    - most significant byte containing the value 12

Address	Value
184	12
185	34
186	56
187	78

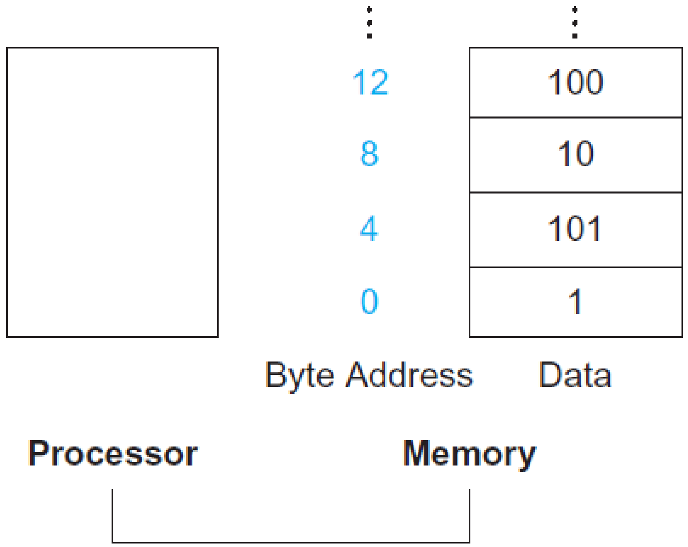
Big endian

Address	Value
184	78
185	56
186	34
187	12

Little endian

Example

- C code: `g = h + A[8];`
  - g in $s1, h in $s2, base address of A in $s3
- Compiled MIPS code:
  - Index 8 requires offset of 32
  - 4 bytes per word



offset

- `lw $t0, 32($s3) # load word`

base register

- `add $s1, $s2, $t0`

Registers vs Memory

- Registers
  - Small set, typically 8 to 32
  - Fast locations for data
  - In MIPS, data must be in registers to perform arithmetic
  - Compiler must use registers for variables as much as possible

- Memory
  - Accessed only by data transfer instructions (ld/st)
  - MIPS uses byte addresses, sequential word accesses differ by 4
  - Memory holds data structures, arrays, and spilled registers (less frequently used variables)

Immediate Operands

- Program will use a constant in an operation
    - e.g. incrementing an index to point to the next element of an array
- Load a constant from memory to use one
    - For example, to add the constant 4 to register $s3:

```
lw $t0, AddrConstant4($s1)    # $t0 = constant 4
add $s3, $s3, $t0             # $s3 = $s3 + $t0
```

- An alternative is to use immediate operands
- To add 4 to register $s3:

```
addi $s3, $s3, 4   # $s3 = $s3 + 4
```

Representing Instructions

- Instructions are encoded in binary
  - Called machine code

- MIPS instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity

MIPS R-Format Instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount
  - funct: function code (extends opcode)

MIPS I-Format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs

Conditional Operations

- Branch to a labeled instruction if a condition is true
  - otherwise, continue sequentially

branch if equal

- `beq rs, rt, L1`
  - if (`rs == rt`) branch to instruction labeled L1

branch if not equal

- `bne rs, rt, L1`
  - if (`rs != rt`) branch to instruction labeled L1
- `j L1`
  - unconditional jump (unconditional branch) to instruction labeled L1
- `beq` and `bne` are called conditional branches

More Conditional Operators

- Useful to see if a variable is less than another variable
  - E.g, a *for* loop may want to test to see if the index variable is less than 0
- Instruction that compares two registers and sets a third register to 1 if the first is less than the second; otherwise, it is set to 0
- The MIPS instruction is called *set on less than,* or `slt`

```
slt $t0, $s3, $s4   # $t0 = 1 if $s3 < $s4, else $t0 = 0
```

`slt rd, rs, rt`

if (rs < rt) rd = 1; else rd = 0;

`slti rt, rs, constant`

if (rs < constant) rt = 1; else rt = 0;

- MIPS compilers use the `slt, slti, beq, bne`, and the fixed value of 0 to create all relative conditions: equal, not equal, less than, less than or equal, greater than, greater than or equal.
- Use in combination with `beq`, `bne`

```
slt $t0, $s1, $s2     # if ($s1 < $s2)
bne $t0, $zero, L     #   branch to L
```

More Addressing

- Branch Addressing
  - Branch instructions specify
    - Opcode, two registers, target address
  - Most branch targets are near branch
    - Forward or backward
  - PC-relative addressing

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Jump Addressing
  - Jump (`j` and `jal`) targets could be anywhere in text segment
    - Encode full address in instruction

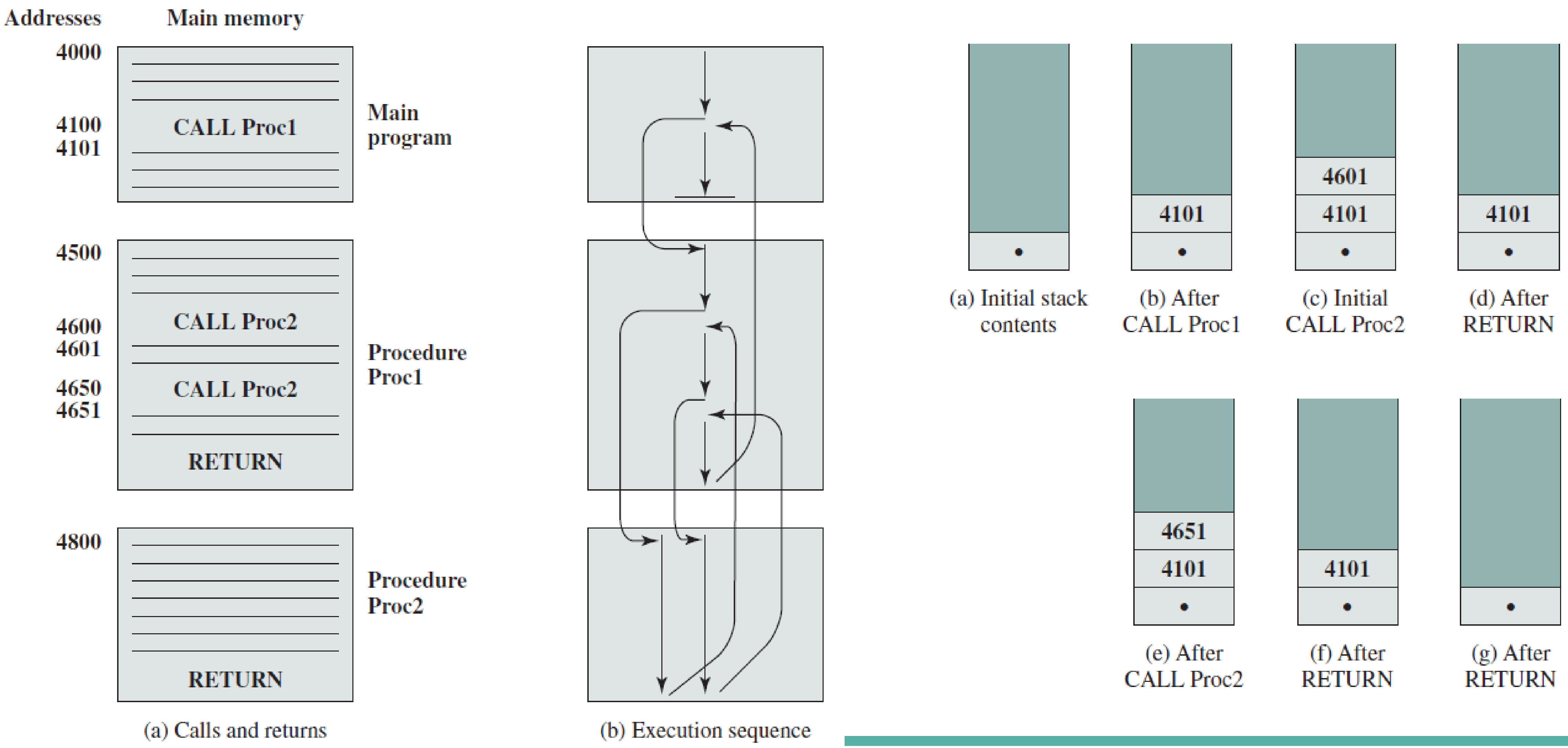
op	address
6 bits	26 bits

Supporting Procedures in Computer HW

- Structure programs and allow code to be reused

- Steps required
  1. Place parameters in registers
  2. Transfer control to procedure
  3. Acquire storage for procedure
  4. Perform procedure’s operations
  5. Place result in register for caller
  6. Return to place of call

Use of Stack to Implement Nested Subroutines

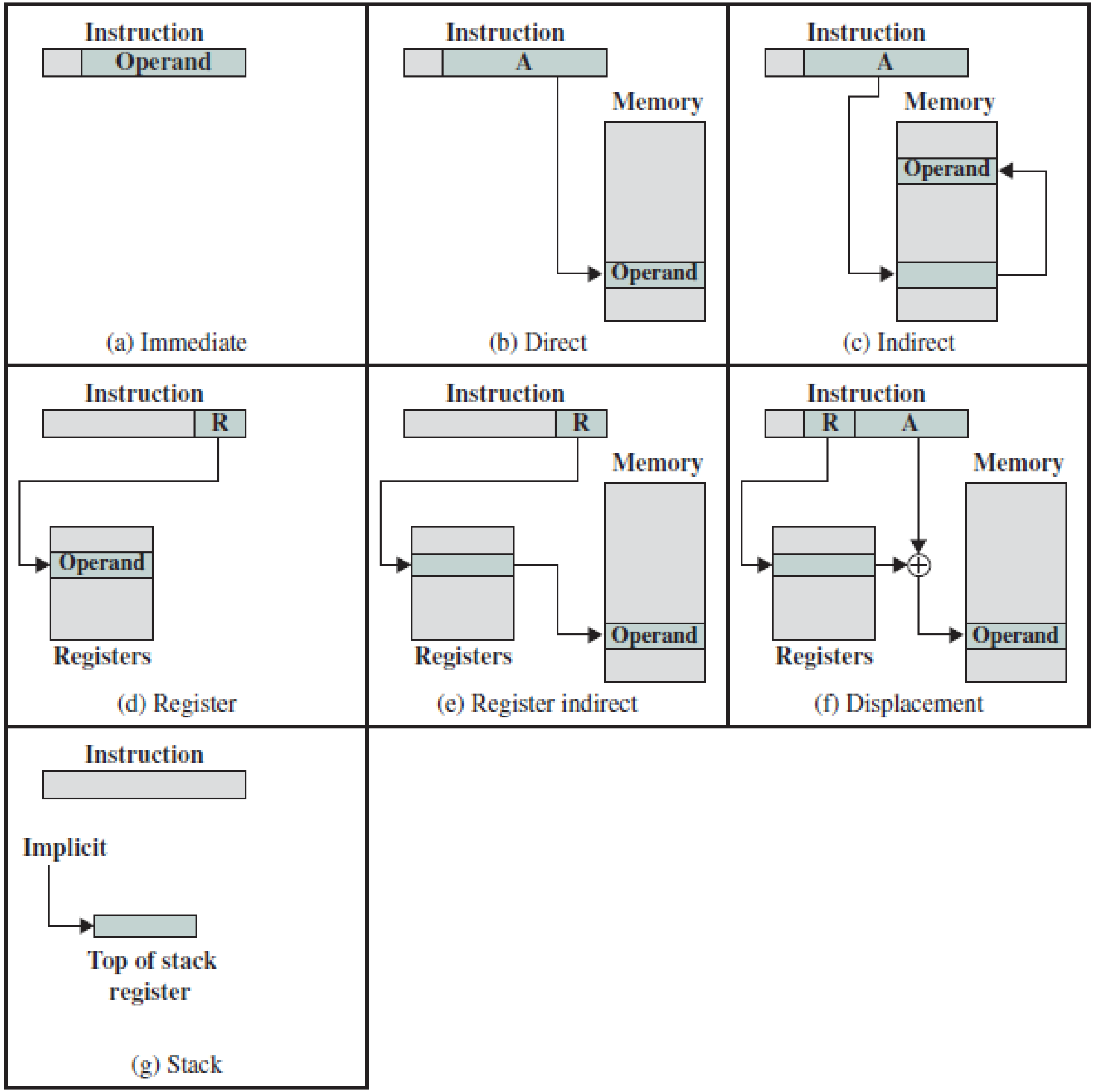


MIPS Design Principles

- **Simplicity favors regularity**
  - Fixed size instructions
  - Small number of instruction formats
  - Opcode always first 6 bits
- **Smaller is faster**
  - Limited instruction set
  - Limited number of registers in register file
  - Limited number of addressing modes
- **Make the common case fast**
  - Arithmetic operands from the register file(load-store architecture)
  - Allow instructions to contain immediate operands
- **Good design demands good compromises**
  - Three instruction formats

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack



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

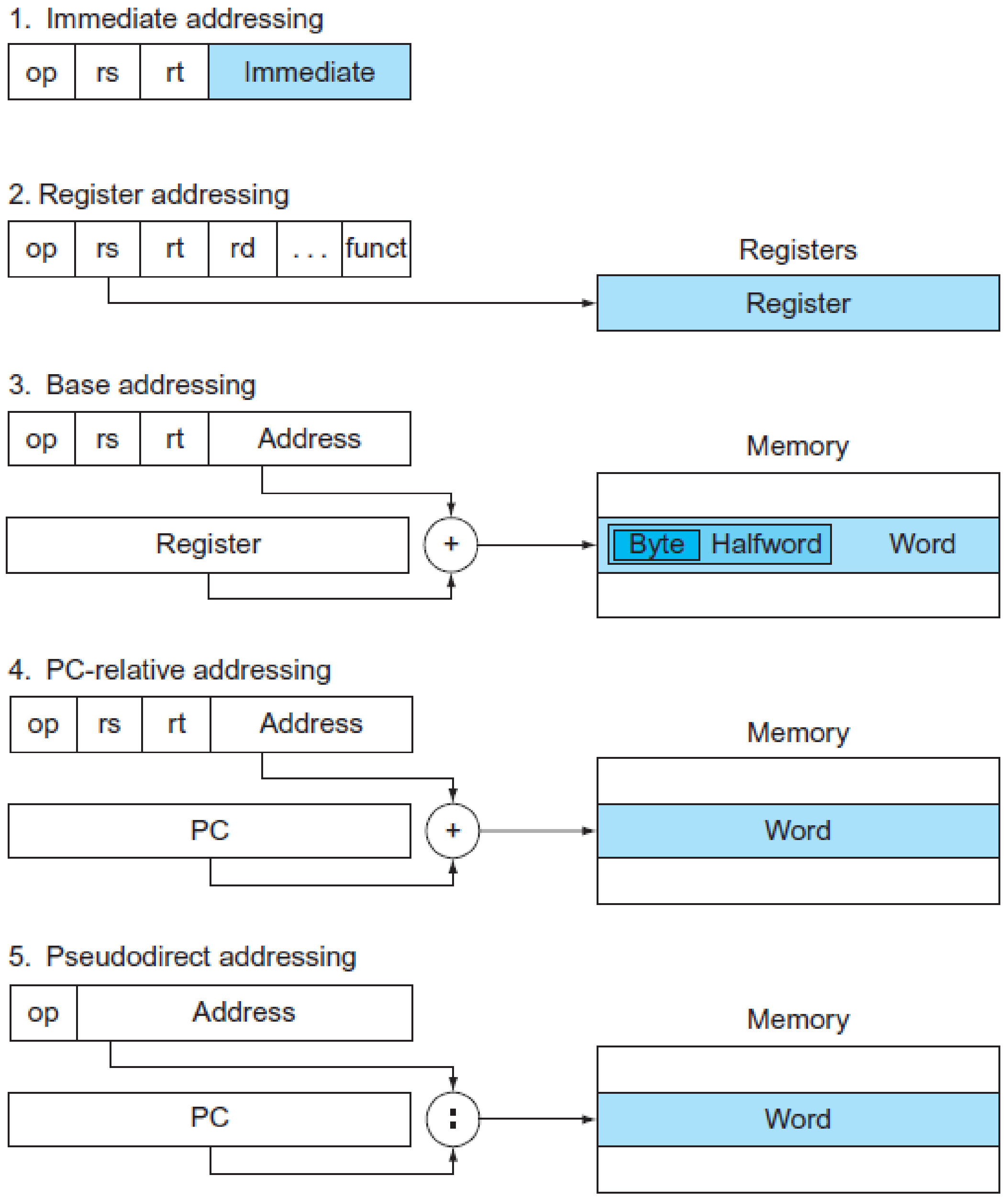
- A = contents of an address field in the instruction
- R = contents of an address field in the instruction that refers to a register
- EA = actual (effective) address of the location containing the referenced operand
- (X) = contents of memory location X or register X

MIPS Addressing Modes

- **Base or displacement addressing**: operand is at the memory location whose address is the sum of a register and a constant in the instruction

- **PC-relative addressing**: branch address is the sum of the PC and a constant in the instruction

- **Pseudodirect addressing**: jump address is the 26 bits of the instruction concatenated with the upper bits of the PC



Variable Length Instructions

- So far have used a single fixed instruction length
- Designer may choose to provide a variety of instruction formats of different lengths
- Makes it easy to provide a large repertoire of opcodes, with different opcode lengths
- Addressing can be more flexible, with various combinations of register and memory references plus addressing modes
- With variable-length instructions, these many variations can be provided efficiently and compactly
- Increase in complexity of processor