Computer Organization and Architecture

14 Instruction Sets

Tongwei Ren

Dec. 3, 2019

NANJING UNIVERSITY

Review

- Bus
  - Concept, characteristic, type, component

- Design Element
  - Bus type: dedicated, multiplexed
  - Arbitration: centralized, distributed
  - Timing: synchronous, asynchronous, semi-synchronous, split bus transaction
  - Bus bandwidth and data transfer rate
  - Bus hierarchy: single bus, double bus, multiple bus

Computer Function

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory

- The processor does the actual work by executing instructions specified in the program

回顾：指令

回顾：指令

- 指令是计算机处理的最基本单位
  - 操作码（指令执行的内容）+操作数（要操作的对象）
- 多周期实现方案
  - 可以将一条指令的执行分解为一系列步骤
   - 取指令，译码/取寄存器，执行/有效地址/完成分支，访问内存，存储结果

Instruction Cycle

- Instruction cycle: The processing required for a single instruction
    - Instruction fetch: fetch one instruction from memory each time
    - Instruction execute: execute each instruction
- Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered
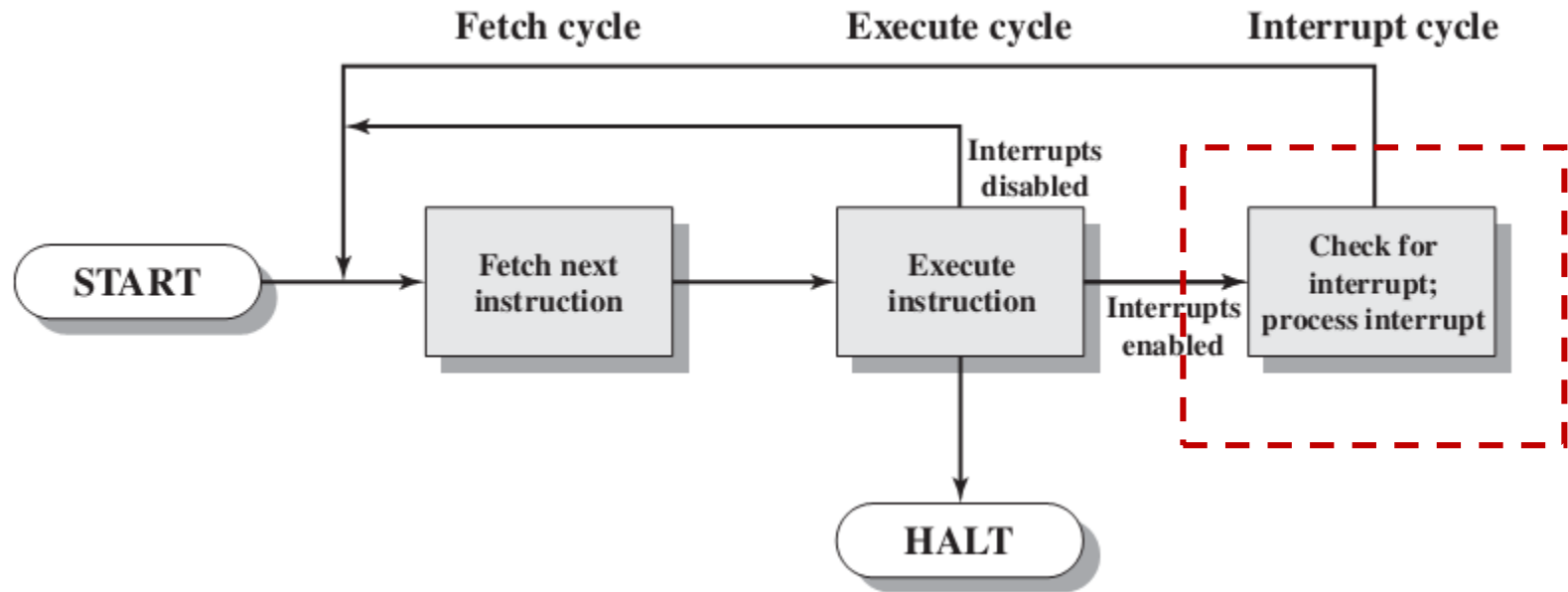
Fetch cycle   **Execute cycle**

START → Fetch next instruction → Execute instruction → HALT
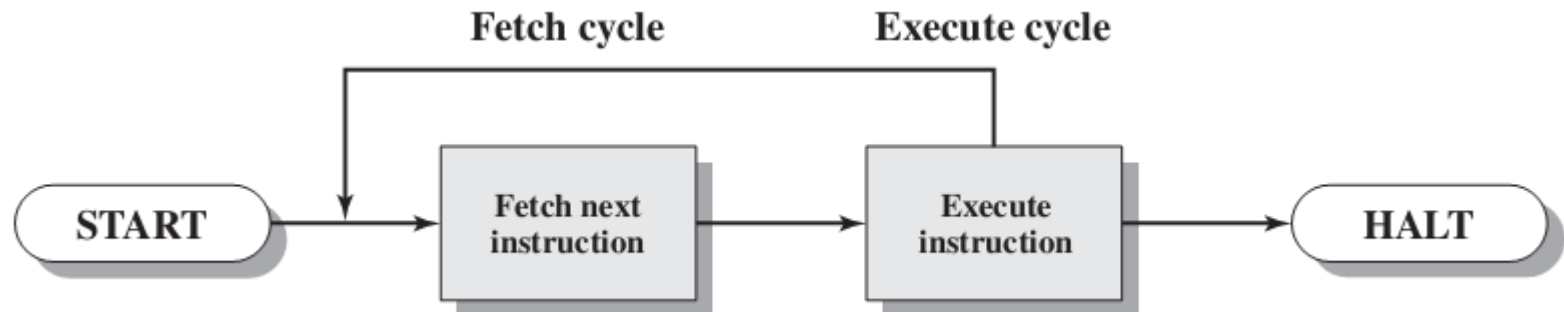
Instruction Cycle (cont.)
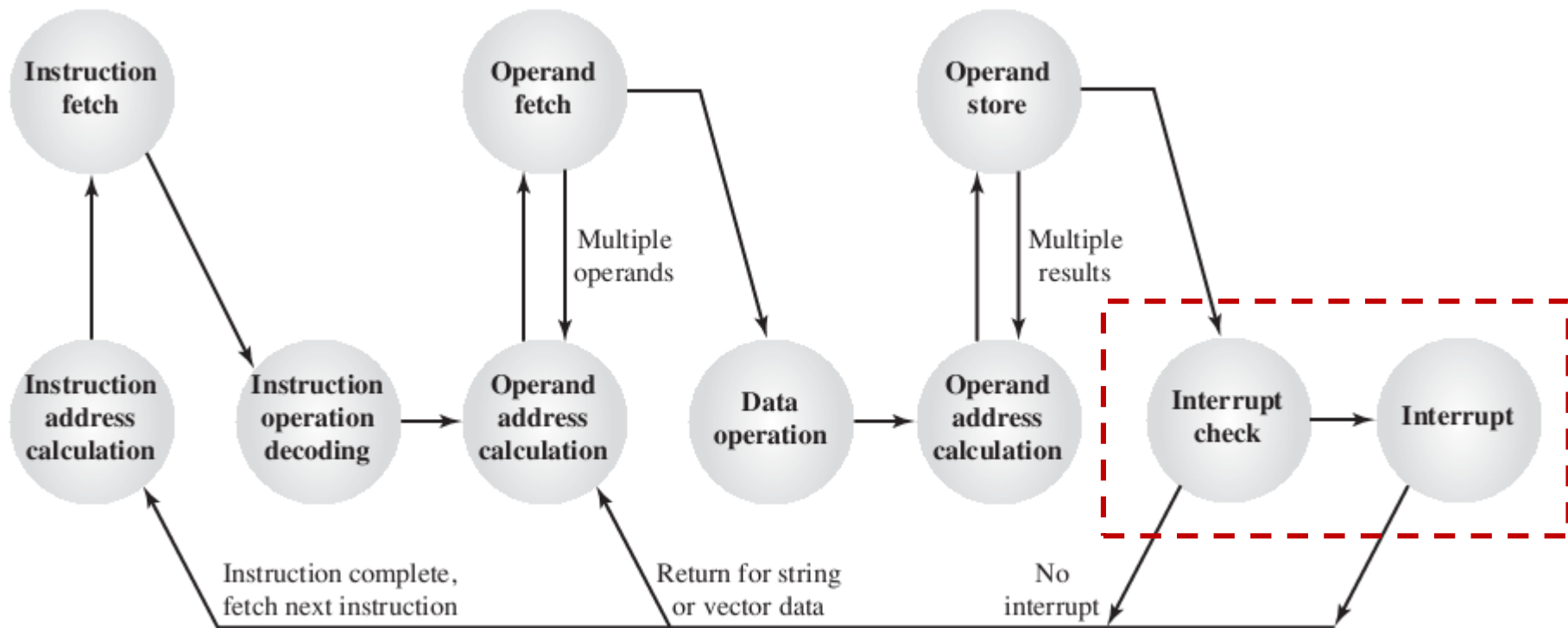
- Instruction cycle state diagram
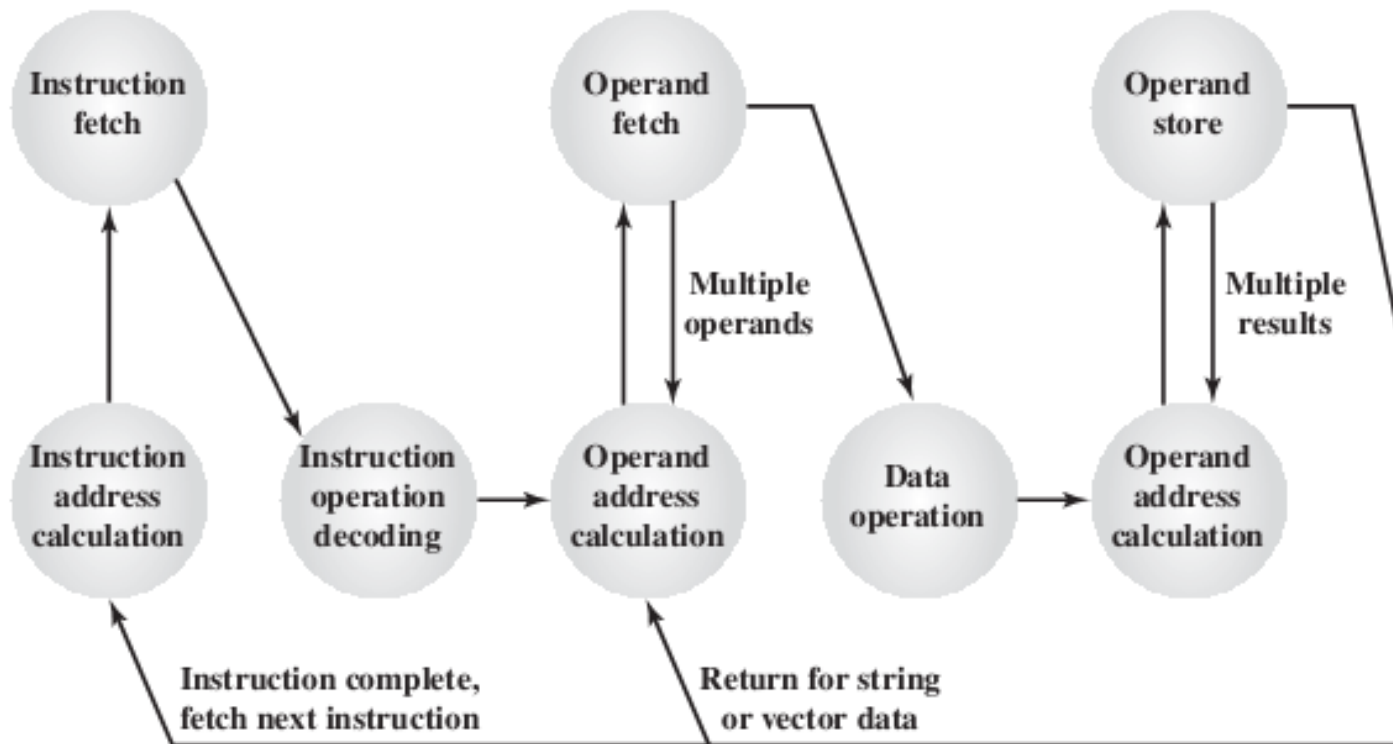
Instruction Cycle With Interrupts

- Instruction Cycle with Interrupts

Instruction Cycle With Interrupts (cont.)

- Instruction cycle state diagram with interrupts

Instruction

- Elements of instruction
  - Operation code: specify the operation to be performed
  - Source operand reference: the operation may involve one or more source operands, that is, operands that are inputs for the operation
  - Result operand reference: the operation may produce a result
  - Next instruction reference: tell the processor where to fetch the next instruction after the execution of this instruction is complete.

Instruction (cont.)

- Instruction representation
    - Each instruction is represented by a sequence of bits
    - **Instruction format**: the instruction is divided into fields, corresponding to the constituent elements of the instruction
    - With most instruction sets, more than one format is used

4 Bits	6 Bits	6 Bits
Opcode	Operand reference	Operand reference

← 16 Bits →
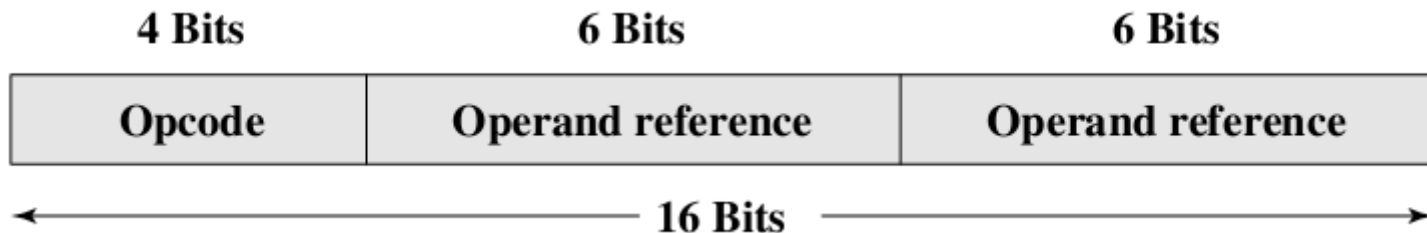
Instruction (cont.)

- Instruction representation (cont.)
  - Symbolic representation: help both the programmer and the reader of textbooks to deal with instructions
  - Opcodes are represented by abbreviations, called mnemonics
    - ADD: add, SUB: subtract, MUL: multiply, DIV: divide, LOAD: load data from memory, STOR: store data to memory…
  - Operands are also represented symbolically
    - Replace operands with the register name or memory address

Operations

- The number of different opcodes varies widely from machine to machine

- The same general types of operations are found on all machines

  - Data transfer

  - Arithmetic

  - Logical

  - Conversion

  - I/O

  - System control

  - Transfer of control

Operations (cont.)

- Data transfer

  - Specify the location of source and destination operands

  - Indicate the length of data to be transferred

  - Specify the mode of addressing for each operand

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

Operations (cont.)

- Arithmetic
  - The execution of an arithmetic instruction may involve data transfer operations to position operands for input to the ALU, and to deliver the output of the ALU

Type	Operation Name	Description
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

Operations (cont.)

- Logical
    - Bit twiddling: manipulating individual bits of a word or other addressable units
    - Shifting and rotating function

Type	Operation Name	Description
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

Operations (cont.)

- Conversion
  - Change the format or operate on the format of data

Type	Operation Name	Description
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)

Operations (cont.)

- I/O

  - A variety of I/O approaches are implemented by only a few I/O instructions, with the specific actions specified by parameters, codes, or command words

| 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

Operations (cont.)

- Transfer of control
  - Branch
  - Skip
  - Procedure call

Type	Operation Name	Description
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

Operations (cont.)

- Transfer of control (cont.)
  - Branch / Jump: has as one of its operands the address of the next instruction to be executed
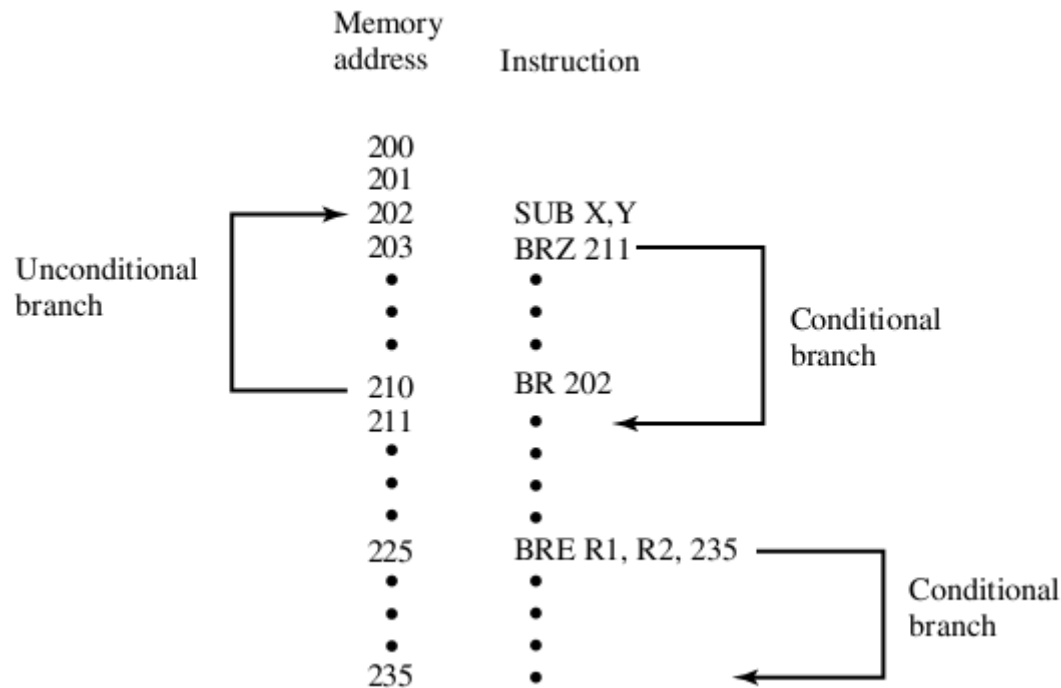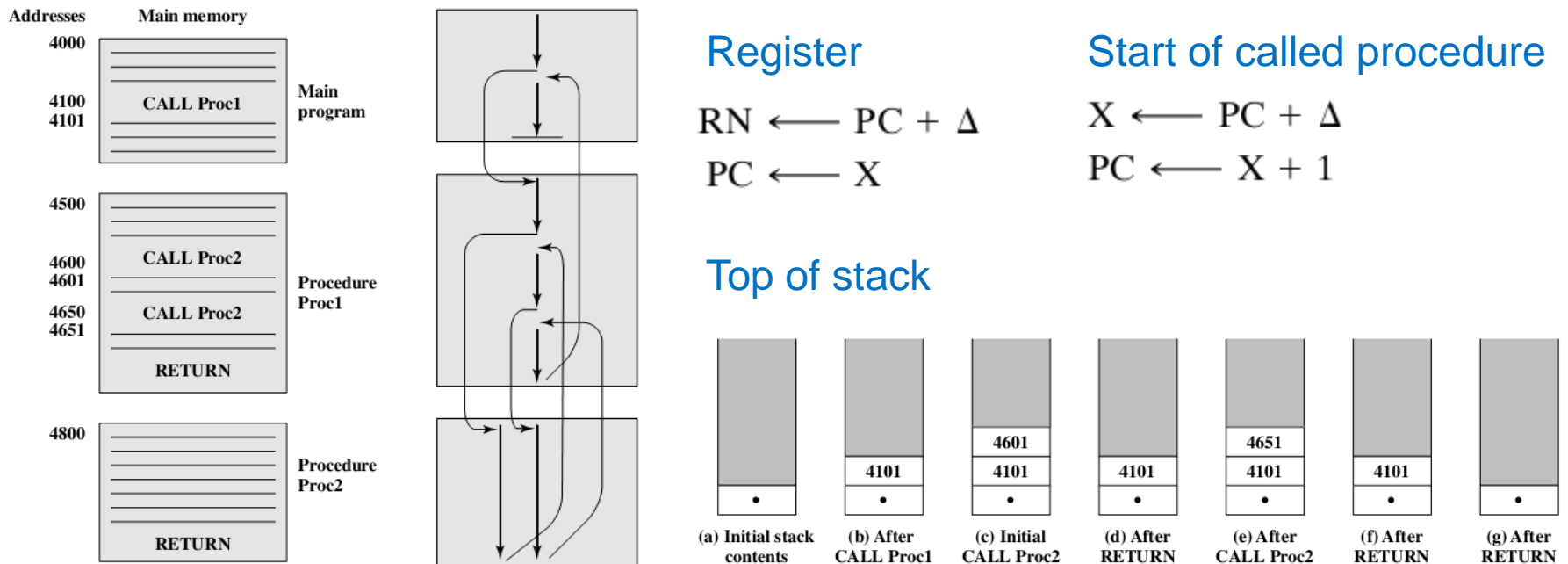
Operations (cont.)

- Transfer of control (cont.)
  - Skip: include an implied address, which equals the address of the next instruction plus one instruction length

```
301
  •
  •
  •
309    ISZ    R1
310    BR     301
311
```

Operations (cont.)

- Transfer of control (cont.)

    - Procedure call: a call instruction that branches from the present location to the procedure, and a return instruction that returns from the procedure to the place from which it was called

Register

$$RN \longleftarrow PC + \Delta$$
$$PC \longleftarrow X$$

Start of called procedure

$$X \longleftarrow PC + \Delta$$
$$PC \longleftarrow X + 1$$

Top of stack

Operands

- The most important general categories of operands are

  - Addresses

  - Numbers

  - Characters

  - Logical data

Operands (cont.)

- Address

    - An instruction is required to contain four address references: two source operands, one destination operand, and the address of the next instruction

    - The address of the next instruction is implicit

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	$Y \leftarrow Y \div T$

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	$Y \leftarrow Y \div T$

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	$Y \leftarrow AC$

Operands (cont.)

- Address
    - Fewer addresses per instruction result in instructions that are more primitive, requiring a less complex processor and shorter instruction length
    - Programs contain more total instructions, which in general results in longer execution times and longer, more complex programs
    - With multiple-address instructions, it is common to have multiple general purpose registers, which allows some operations to be performed solely on registers and makes program execution faster

Operands (cont.)

- Numbers
  - Numbers stored in a computer are limited
    - Limitation on the magnitude of numbers
    - Limitation on the precision of floating-point numbers
  - Types of numerical data
    - Binary integer or binary fixed point
    - Binary floating point
    - Decimal

Operands (cont.)

- Characters

  - International reference alphabet (IRA) / American standard code for information interchange (ASCII): 7 bits

  - Extended Binary Coded Decimal Interchange Code (EBCDIC): 8 bits

  - Unicode: 16 bits / 32 bits

Operands (cont.)

- Logical data
  - Consider an n-bit unit as consisting of n 1-bit items of data, each item having the value 0 or 1
    - Store an array of Boolean or binary data items, in which each item can take on only the values 1 (true) and 0 (false)
    - There are occasions when we wish to manipulate the bits of a data item

Operands (cont.)

- Big endian ordering and little endian ordering

  - Suppose we have the 32-bit hexadecimal value 12345678 and that it is stored in a 32-bit word in byte-addressable memory at byte location 184

Address	Value
184	12
185	34
186	56
187	78

Big endian

Address	Value
184	78
185	56
186	34
187	12

Little endian

Operands (cont.)

- Big endian ordering and little endian ordering (cont.)

  - Each data item has the same address in both schemes

  - Within any given multibyte scalar value, the ordering of bytes in the little-endian structure is the reverse of that for the big-endian structure

  - Endianness does not affect the ordering of data items within a structure

Operands (cont.)

- Big endian ordering and little endian ordering (cont.)

    - Big endian

        - Character-string sorting

        - Decimal / IRA dumps

        - Consistent order

    - Little endian

        - Easier to converts a 32-bit integer address to a 16-bit integer address

        - Easier to perform higher-precision arithmetic

Operand Reference Format

- An operand reference in an instruction
    - Actual value of the operand
    - Address of the operand
        - Register
        - Main memory / virtual memory
        - …

Addressing Modes

- Immediate

- Direct

- Indirect

- Register

- Register indirect

- Displacement

- Stack

Notation

- 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

- Mode: operand value is present in the instruction

- Usage: define and use constants or set initial values of variables

- Algorithm: Operand = A

- Advantage: no memory reference other than the instruction fetch is required to obtain the operand

- Disadvantage: the size of the number is restricted to the size of the address field
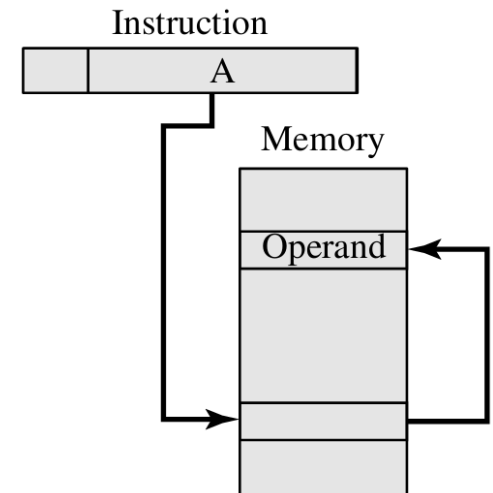
Instruction
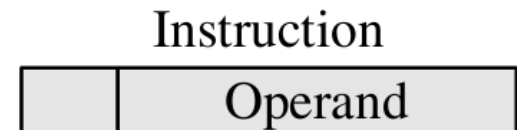
	Operand

Direct Addressing

- Mode: Address field contains the effective address of operand

- Usage: common in earlier generations of computers but is not common on contemporary architectures

- Algorithm: EA = A

- Advantage: only one memory reference and no special calculation
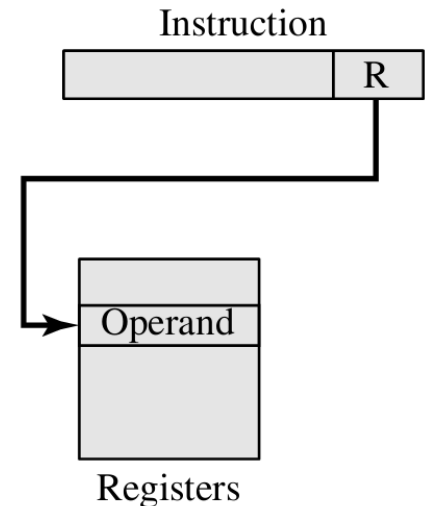
- Disadvantage: limited address space
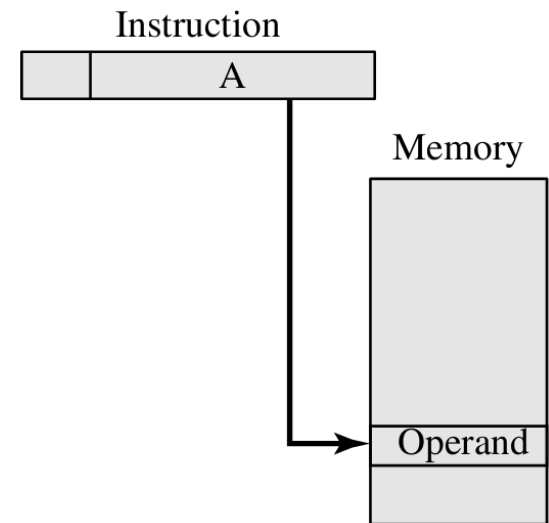
Indirect Addressing

- Mode: Address field refer to the address of a word in memory, which in turn contains a full-length address of operand

- Algorithm: EA = (A)

- Advantage: enlarge address space

- Disadvantage: require two memory references to fetch the operand

- Comment: limitation of the number of address reference can be an asset

Register Addressing

- Mode: address field refers to a register

- Algorithm: EA = R

- Advantage: only a small address field is needed in the instruction, and no time-consuming memory references are required

- Disadvantage: address space is very limited

- Comment: the usage of registers makes sense only if they are employed efficiently
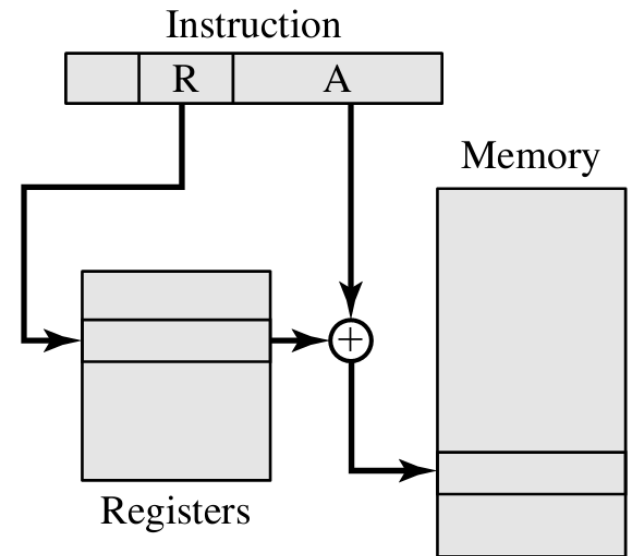
Register Indirect Addressing

- Mode: address field refers to a register

- Algorithm: EA = (R)

- Advantage: enlarge address space, and use one less memory reference than indirect addressing

- Disadvantage: require one more memory reference
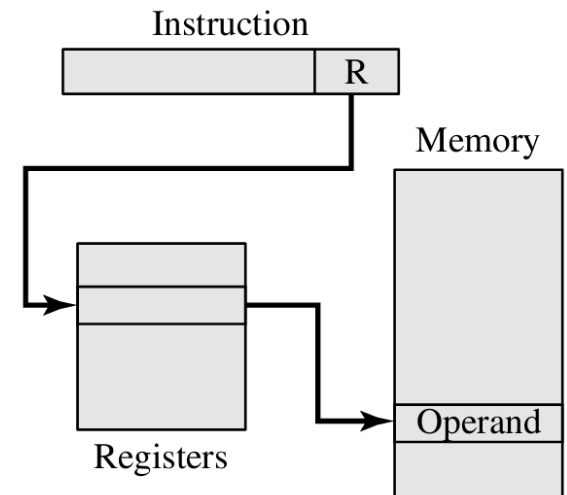
Displacement Addressing

- Mode: combine the capabilities of direct addressing and register indirect addressing

- Algorithm: EA = (R) + A

- Type
  - Relative addressing
  - Base-register addressing
  - Indexing

- Comment: requires that the instruction have two address fields, at least one of which is explicit

Displacement Addressing (cont.)

- Relative addressing
    - Mode: the implicitly referenced register is the program counter (PC), i.e., the next instruction address is added to the address field to produce the EA
    - Usage: floating of public subroutine or relative transfer
    - Algorithm: EA = (PC) + A
    - Advantage: exploits the concept of program locality, and save address bits in the instruction

Displacement Addressing (cont.)

- Base-register addressing

  - Mode: 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

  - Algorithm: EA = (B) + A

 - Usage: program relocation in virtual memory space

Displacement Addressing (cont.)

- Indexing
  - Mode: The address field references a main memory address, and the referenced register contains a positive displacement from that address
  - Algorithm: EA = A + (B)
  - Usage: provide an efficient mechanism for performing iterative operations
  - Extension: combine indirect addressing and indexing
    - Pre-indexing: EA = (A + (R))
    - Post-indexing: EA = (A) + (R)

Stack Addressing

- Mode: The stack pointer is maintained in a register. So references to stack locations in memory are in fact register indirect addresses.

- Comment: Associated with the stack is a pointer references the top of the stack or the third element of the stack (the top two elements of the stack may be in processor registers)

Instruction

Implicit

Top of stack register
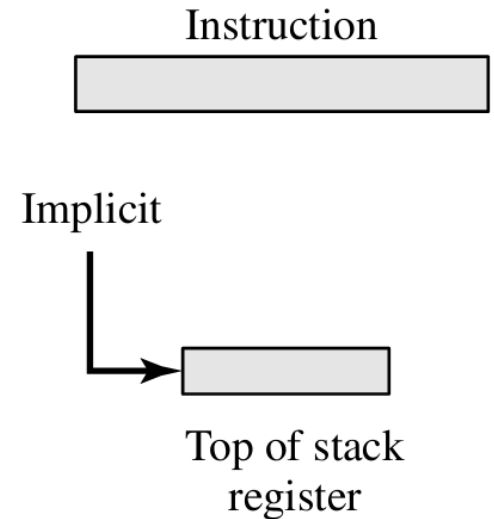
Instruction Set Design

- The instruction set defines many of the functions performed by the processor and thus has a significant effect on the implementation of the processor

- Programmer requirements must be considered in designing the instruction set

Instruction Set Design (cont.)

- Fundamental design issues

    - 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

Instruction Format

- An instruction format defines the layout of the bits of an instruction, in terms of its constituent fields

- An instruction format must include an opcode and, implicitly or explicitly, zero or more operands

- The format must, implicitly or explicitly, indicate the addressing mode for each operand

- For most instruction sets, more than one instruction format is used

Instruction Length

- The most obvious trade-off here is between the desire for a powerful instruction repertoire and a need to save space

    - Programmers want more opcodes, more operands, more addressing modes, and greater address range

    - Longer instruction length may be wasteful

- Either the instruction length should be equal to the memory-transfer length or one should be a multiple of the other

- Shorter instruction can reduce transfer time

- Instruction length should be a multiple of the character length and of the length of fixed-point numbers

Allocation of Bits

- For a given instruction length, there is clearly a trade-off between the number of opcodes and the power of the addressing capability

- Use of variable-length opcodes
  - Use a minimum opcode length but, for some opcodes, additional operations may be specified by using additional bits in the instruction

Allocation of Bits (cont.)

- Factors of addressing bits

  - Number of addressing modes

  - Number of operands

  - Register versus memory: The more that registers can be used for operand references, the fewer bits are needed

  - Number of register sets: for a fixed number of registers, a functional split requires fewer bits to be used in the instruction

  - Address range

  - Address granularity

Variable-Length Instructions

- Provide a variety of instruction formats of different lengths

- Advantage

  - 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

- Disadvantage: increase the complexity of processor

  - Fetch a number of bytes or words equal to at least the longest possible instruction

Summary

- Instruction cycle, Instruction

- Operations, Operands

- Instruction set design

- Addressing modes
  - Immediate, direct, indirect, register, register indirect, displacement, stack

- Instruction format
  - Instruction length, allocation of bits, variable-length instructions

Thank You

rentw@nju.edu.en