Computer Organization

Instruction Set Characteristics, Instruction Formats, Addressing Modes, RTL & Micro-Operations, CISC, RISC.

Chapter (10)

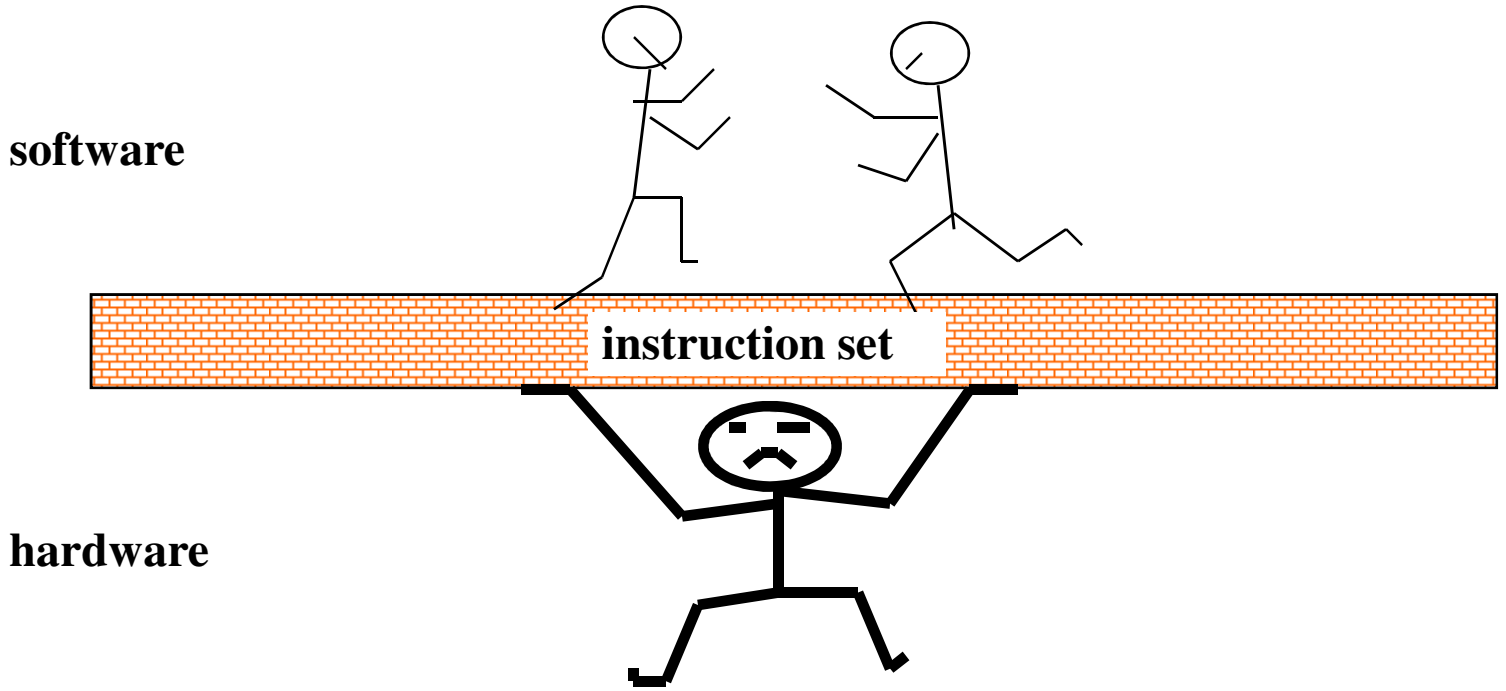
Instruction Set Architecture (ISA)

- Complete set of instructions used by a machine
- Abstract interface between the HW and lowest-level SW.
- An ISA includes the following …
  - Instructions and Instruction Formats
    - Data Types, Encodings, and Representations
    - Programmable Storage: Registers and Memory
    - Addressing Modes: to address Instructions and Data
    - Handling Exceptional Conditions (like division by zero)
- Examples (Versions) First Introduced in
  - Intel (8086, 80386, Pentium, ...) 1978
  - MIPS (MIPS I, II, III, IV, V) 1986
  - PowerPC (601, 604, …) 1993

The Instruction Set Architecture

- ISA is considered part of the SW
- Must be designed to survive changes in hardware technology, software technology, and application characteristic.
  - Is the agreed-upon interface between all the software that runs on the machine and the hardware that executes it.
- Advantages:
  - Different implementations of the same architecture
  - Easier to change than HW
  - Standardizes instructions, machine language bit patterns, etc.
- Disadvantage:
  - Sometimes prevents using new innovations

Instruction Set Architecture: Critical Interface



- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

Intel 8086 instruction set

- There were **116 instructions** in the **Intel 8086** instruction set

Complete 8086 instruction set

	CMPSB				MOV		
AAA	CMPSW	JAE	JNBE	JPO	MOVSB	RCR	SCASB
AAD	CWD	JB	JNC	JS	MOVSW	REP	SCASW
AAM	DAA	JBE	JNE	JZ	MUL	REPE	SHL
AAS	DAS	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DEC	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DIV	JE	JNL	LEA	NOT	REPZ	STD
AND	HLT	JG	JNLE	LES	OR	RET	STI
CALL	IDIV	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IMUL	JL	JNP	LODSW	POP	ROL	STOSW
CLC	IN	JLE	JNS	LOOP	POPA	ROR	SUB
CLD	INC	JMP	JNZ	LOOPE	POPF	SAHF	TEST
CLI	INT	JNA	JO	LOOPNE	PUSH	SAL	XCHG
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA	SAR	XLATB
CMP	IRET	JNB	JPE	LOOPZ	PUSHF	SBB	XOR
	JA				RCL		

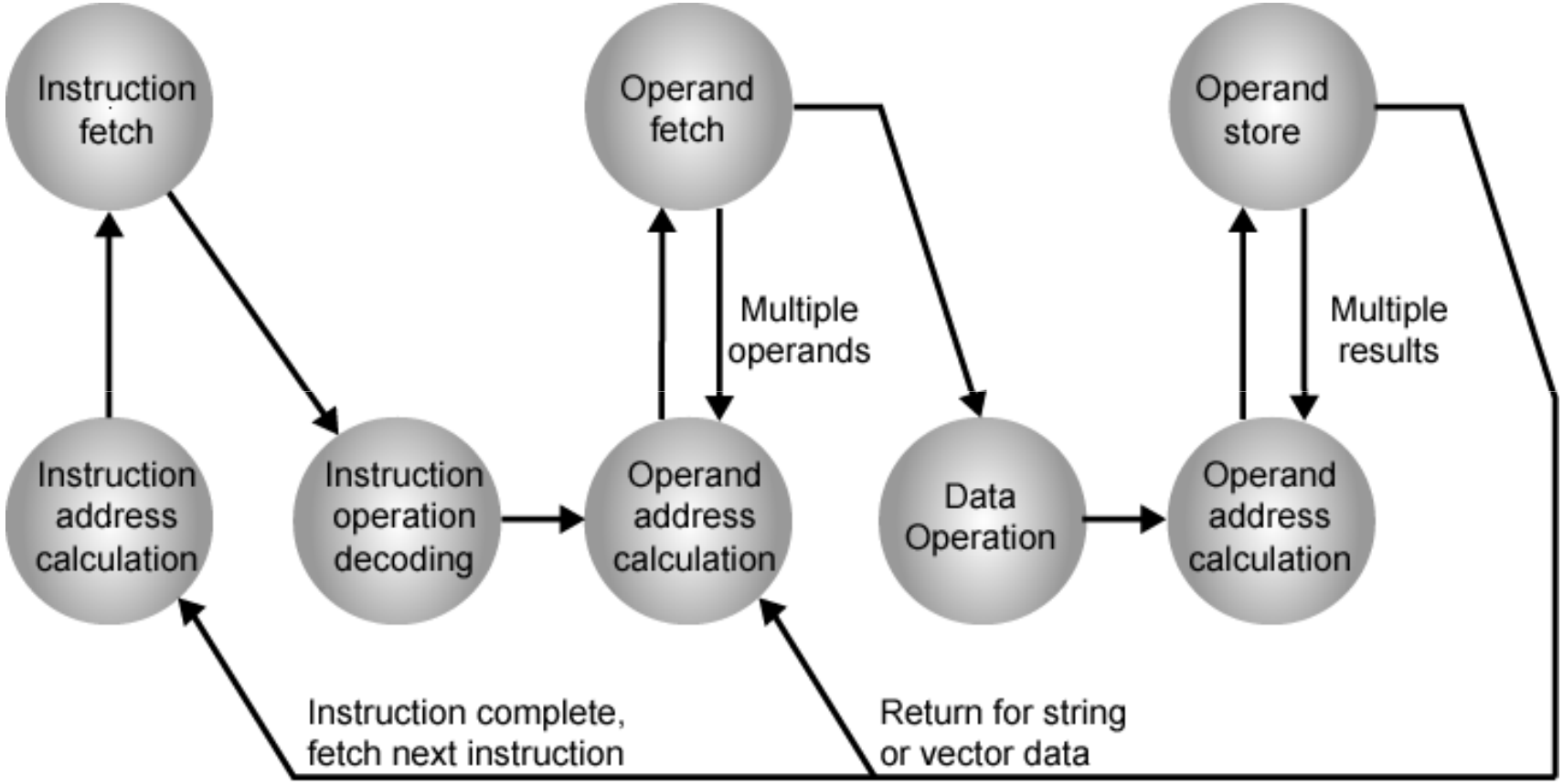
Elements of an Instruction

- Operation code (Op code)
  - —Specify the operation (e.g., ADD, I/O)
- Source Operand reference
  - —Operands that are input to the operation.
- Result Operand reference
  - —Put the answer here
- Next Instruction Reference
  - —Tells the processor where to fetch the next instruction

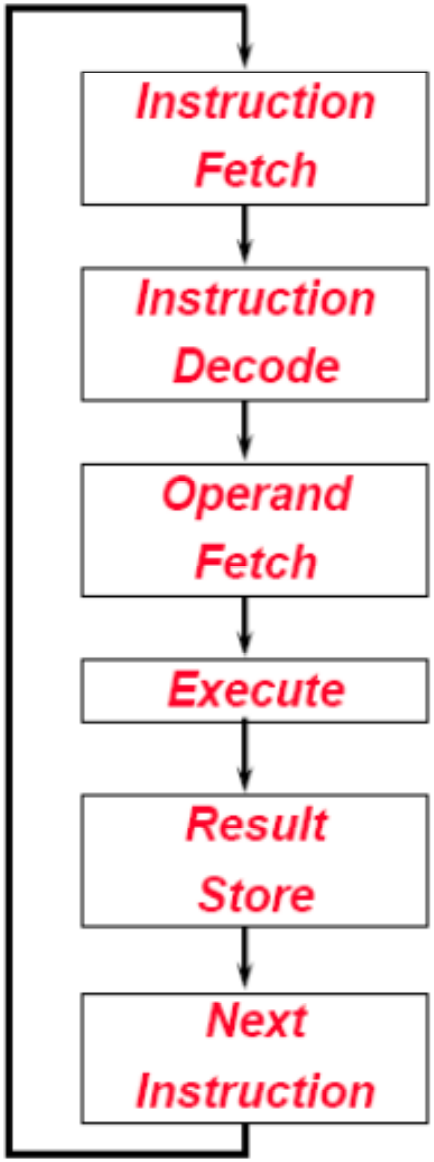
Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

Instruction Cycle State Diagram



Generic CPU Machine Instruction Execution Steps



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use (if required)

Determine successor instruction

Where have all the Operands Gone?
Where is the next instruction to be fetched?

- Main memory (or virtual memory or cache)
- CPU register
- I/O device

Typical Operations

Data Movement	Load (from memory) Store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	Data Types: (signed & unsigned) Integer (binary + decimal) (signed & unsigned) Floating Point Numbers Operations: Add, Subtract, Multiply, Divide
Logical	Not, and, or, set, clear
Shift	Arithmetic (& Logical) shift (left/right), rotate (left/right)
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronisation	test & set (atomic r-m-w)
String	search, compare, translate

Types of Operand

- Addresses
- Numbers
  - —Integer/floating point
- Characters
  - —ASCII etc.
- Logical Data
  - —Bits or flags

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Data Transfer

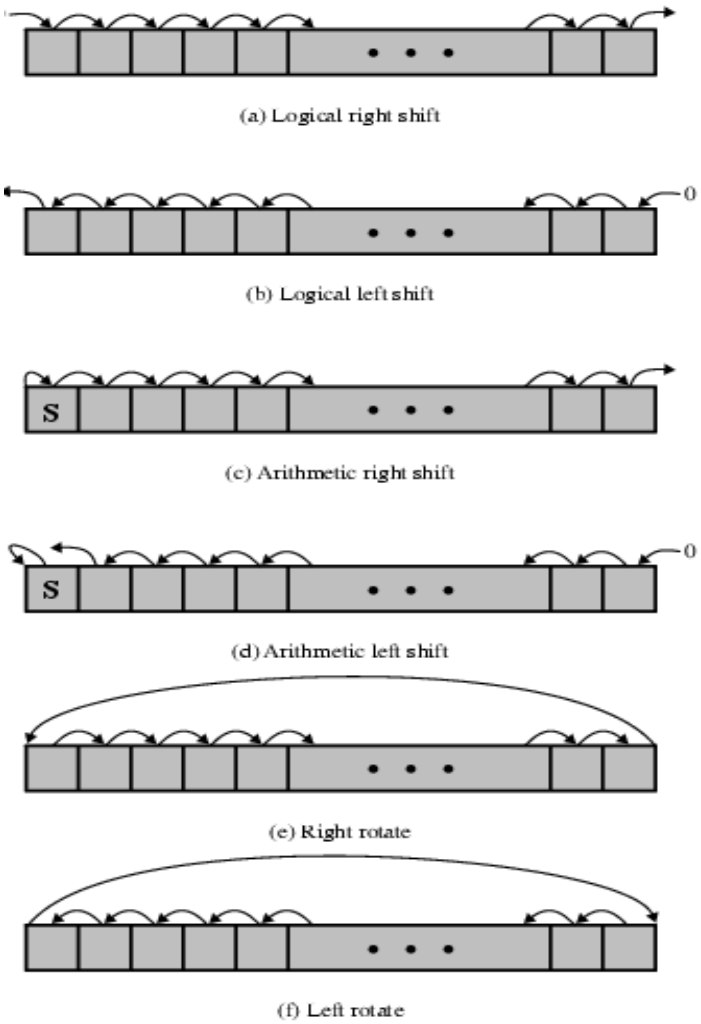
- Specify
  - Source
  - Destination
  - Amount of data
- May be different instructions for different movements
  - e.g. IBM 370
- Or one instruction and different addresses
  - e.g. VAX

Arithmetic

- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Types of Operation

Shift and Rotate Operations



Logical and Conversion

- Bitwise operations
- AND, OR, NOT
- E.g. Binary to Decimal

Types of Operation

- Input/Output
  - May be specific instructions
  - May be done using data movement instructions (memory mapped)
  - May be done by a separate controller (DMA)
- Systems Control
  - For operating systems use

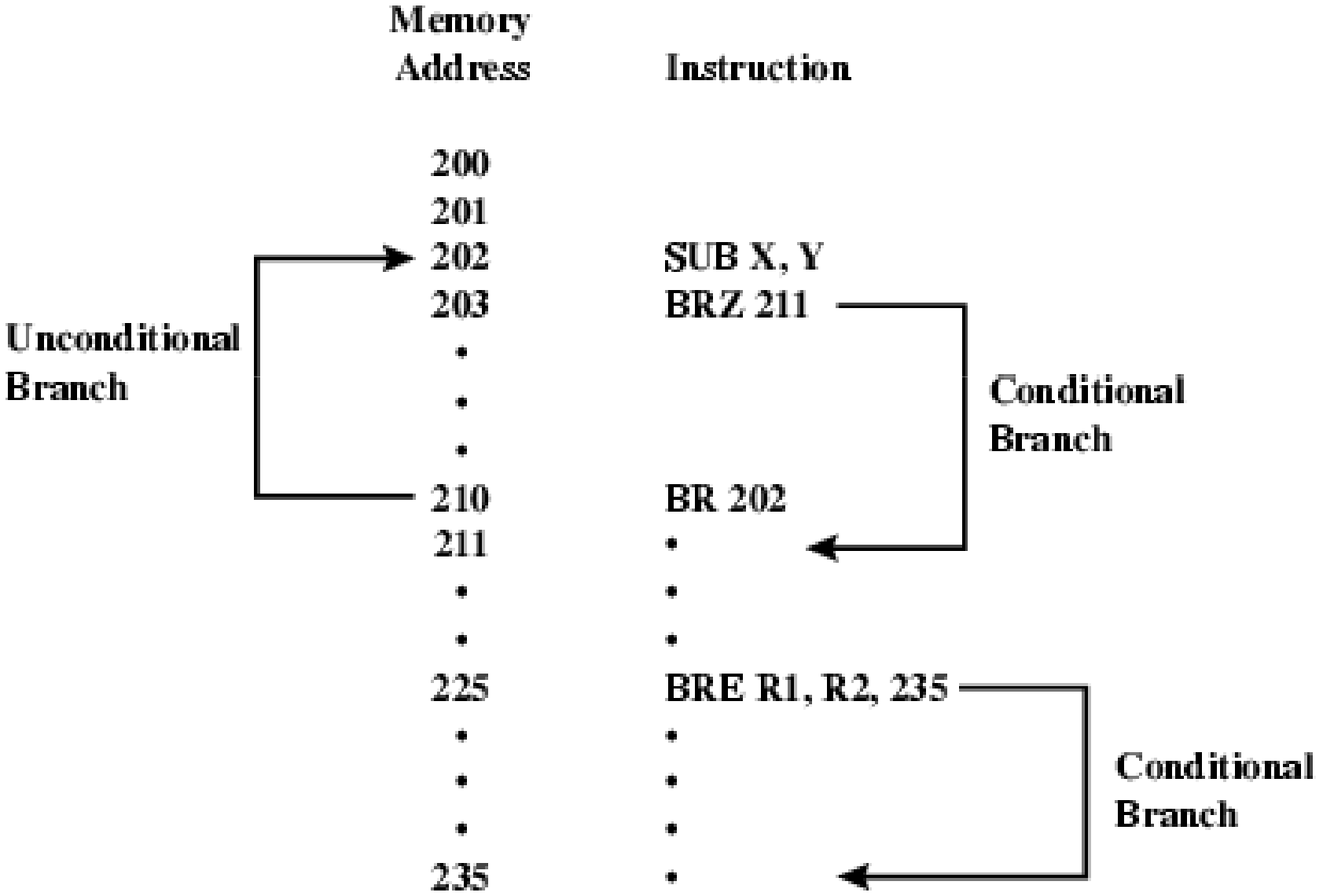
Transfer of Control

- Branch
  - e.g. ***BRZ X*** branch to x if result of (ADD,SUB,…) is zero
  - See next slide
- Skip
  - e.g. increment and skip if zero ISZ

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

  * eg. R1is set to -1000, the loop will be executed 1000 times
- Subroutine call
  - c.f. interrupt call

Branch Instruction



Procedure Calls Instructions

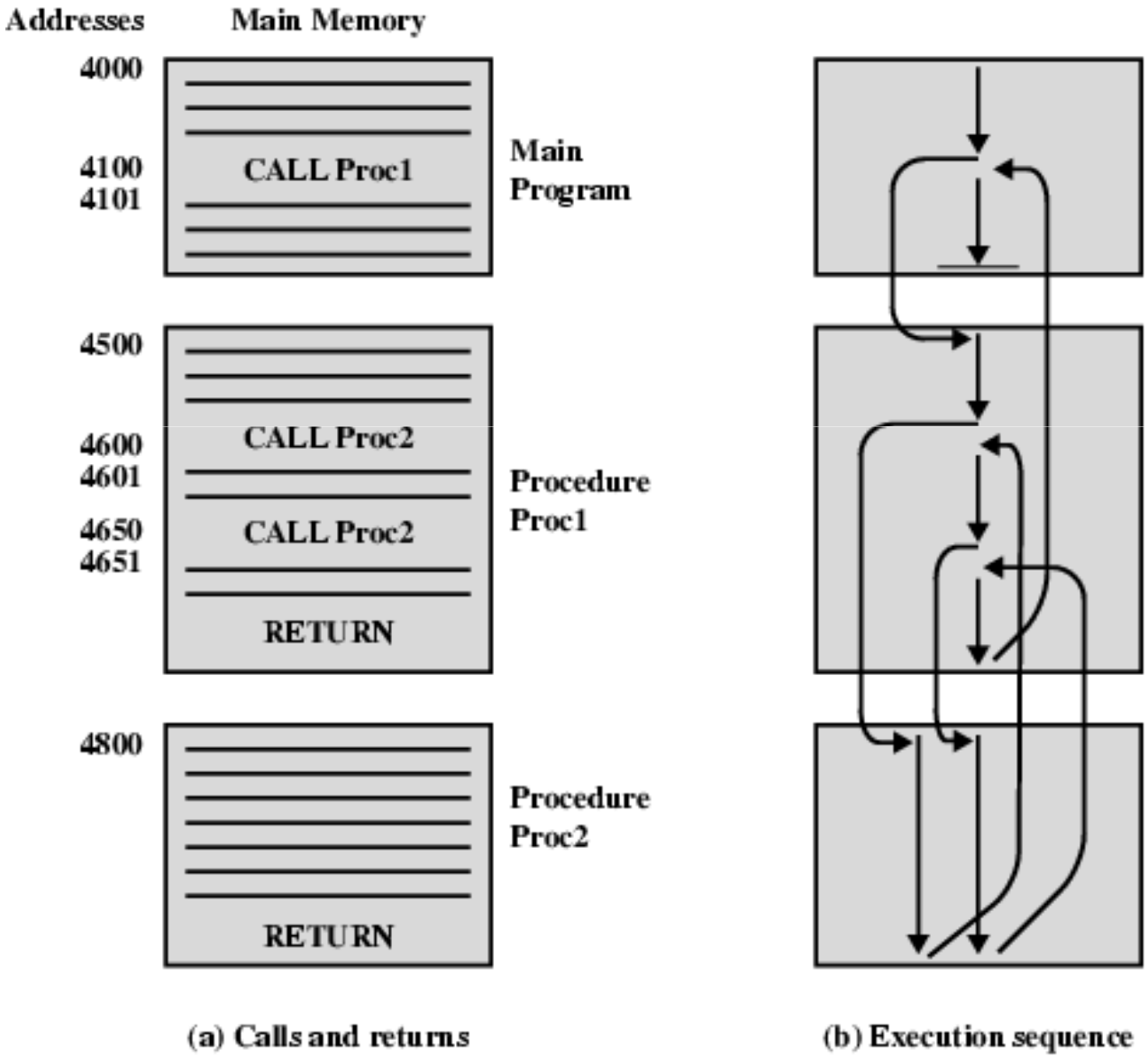
- Computer program that is incorporated with larger program.
- At any point in the program the procedure may be invoked, or ***called***
- When the procedure is executed, return to the point at which the call took place.
- Advantages:
  - Economy:
    - The same piece of Code can be used many times- **efficient use of storage space in the system**
  - Modularity
    - Allow large programming tasks to be divided into smaller units which **eases the programming task**

Procedure Calls Instructions

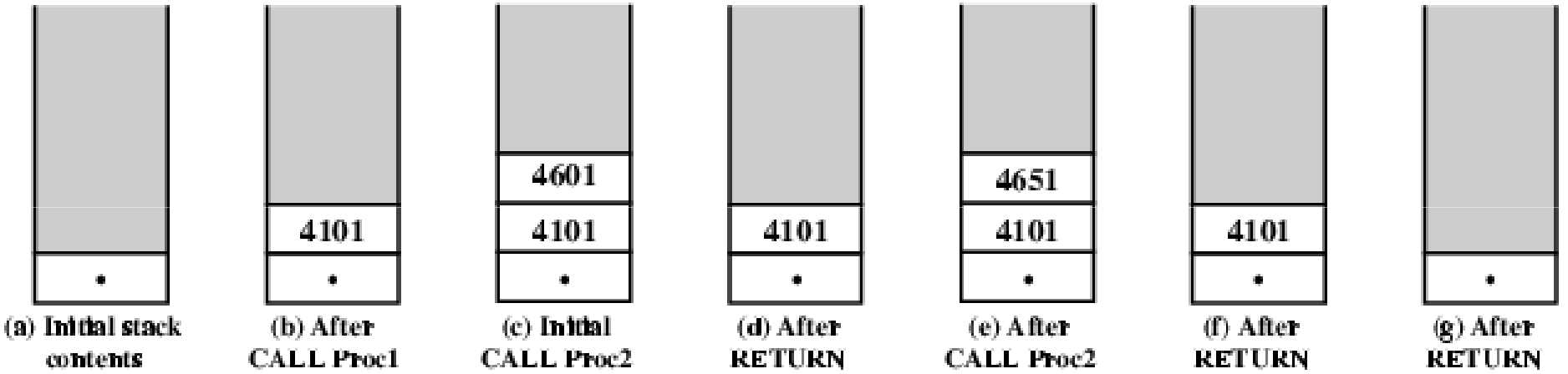
- Involves two basic instructions
  - —Call: branch to the procedure location
  - —Return: from the procedure to the place from which it was called

- ***Stack*** can be used to store the return address.

Nested Procedure Calls



Use of Stack


4101
4601
4101
4101
4651
4101
4101
(a) Initial stack contents
(b) After CALL Proc1
(c) Initial CALL Proc2
(d) After RETURN
(e) After CALL Proc2
(f) After RETURN
(g) After RETURN

Number of Addresses (a)

- # of addresses contained in each instruction
  - —May be 1, 2, 3 or 4 addresses
- 3 addresses
  - —Operand 1, Operand 2, Result
  - —ADD a,b,c (a = b + c;)
- 4 addresses
  - —Operand 1, Operand 2, Result, and next instruction
  - —Not common
  - —Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - ADD a,c (a = a + b)
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results
- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - ADD B (AC = AC + B)
  - Common on early machines

REVERSE POLISH NOTATION

Arithmetic Expressions: A + B

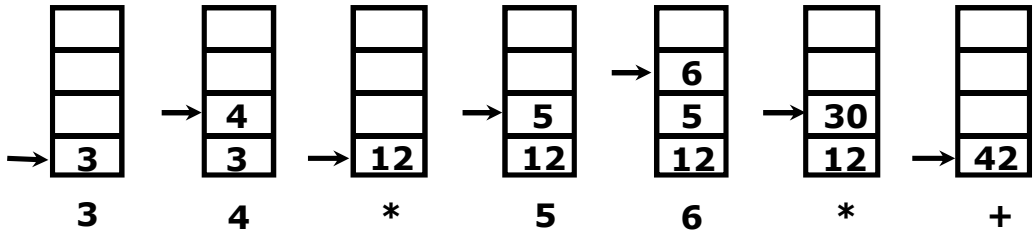
A + B Infix notation
+ A B Prefix or Polish notation
A B + Postfix or reverse Polish notation

- The reverse Polish notation is very suitable for stack manipulation

Evaluation of Arithmetic Expressions

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



Number of Addresses (d)

- 0 (zero) addresses
  - —Applicable to a special memory organization called ***Stack***
  - —Stack is known location
  - —Often at least the top two stack elements are in processor registers
    - ADD
    - All addresses implicit
  - —e.g. push a
  - — push b
  - — add
  - — pop c
  - —c = a + b

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions
- Most processor designs involve a variety of instruction formats.

Fundamental Issues in Instruction Set Design

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
  - The data type that the processor can deal with
  - E.g., Pentium can deal wit data types of:
    - Byte, 8 bits
    - Word, 16 bits
    - Doubleword, 32 bits
    - Quadword, 64 bits
    - Other data type…
- Instruction formats
  - Length of op code field
  - Number of addresses

Fundamental Issues in Instruction Set Design

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)
- RISC v CISC

Exercise For Students --- Next Sat. 23/9

- Find out about instruction set for Pentium and PowerPC
- Start with Stallings
- Visit web sites

Byte Order
(A portion of chips?)

- What order do we read numbers that occupy more than one byte
- e.g. (numbers in hex to make it easy to read)
- 12345678 can be stored in 4x8bit locations as follows

Byte Order (example)

Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
186	78	12

- i.e. read top down or bottom up?

Byte Order Names

- The problem is called Endian
- The system on the left has the least significant byte in the lowest address
- This is called big-endian
- The system on the right has the least significant byte in the highest address
- This is called little-endian

Example of C Data Structure

```
struct{
   int      a;     //0x1112_1314                    word
   int      pad;   //
   double   b;     //0x2122_2324_2526_2728          doubleword
   char*    c;     //0x3132_3334                    word
   char     d[7];  //'A','B','C','D','E','F','G'   byte array
   short    e;     //0x5152                         halfword
   int      f;     //0x6161_6364                    word
} s;
```

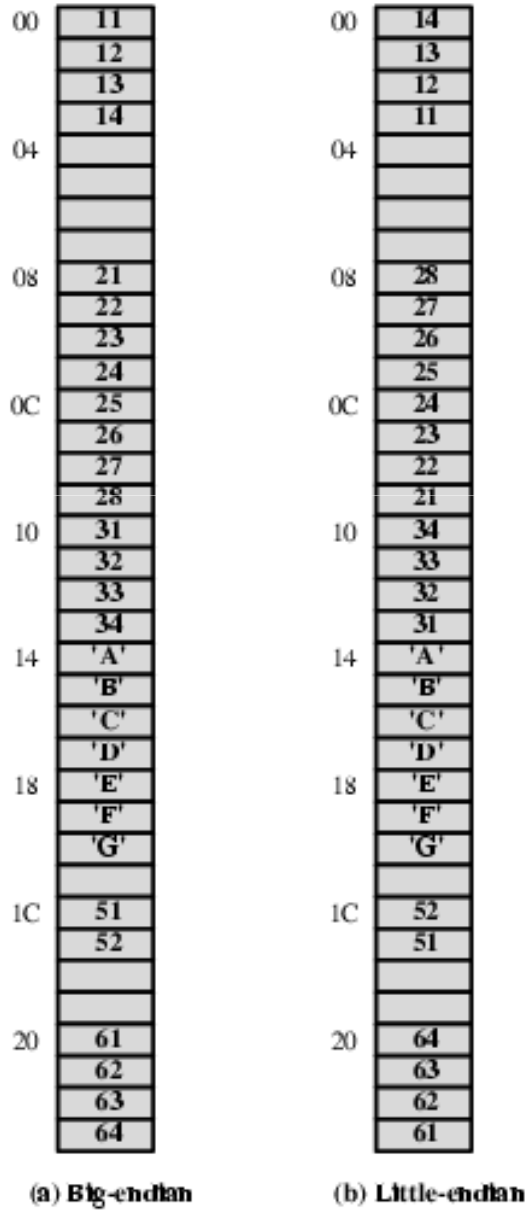
Big-endian address mapping

Byte Address								
00	**11** 00	**12** 01	**13** 02	**14** 03	04	05	06	07
08	**21** 08	**22** 09	**23** 0A	**24** 0B	**25** 0C	**26** 0D	**27** 0E	**28** 0F
10	**31** 10	**32** 11	**33** 12	**34** 13	**'A'** 14	**'B'** 15	**'C'** 16	**'D'** 17
18	**'E'** 18	**'F'** 19	**'G'** 1A	1B	**51** 1C	**52** 1D	1E	1F
20	**61** 20	**62** 21	**63** 22	**64** 23				

Little-endian address mapping

								Byte Address
00	07	06	05	04	**11** 03	**12** 02	**13** 01	**14** 00
08	**21** 0F	**22** 0E	**23** 0D	**24** 0C	**25** 0B	**26** 0A	**27** 09	**28** 08
10	**'D'** 17	**'C'** 16	**'B'** 15	**'A'** 14	**31** 13	**32** 12	**33** 11	**34** 10
18	1F	1E	**51** 1D	**52** 1C	1B	**'G'** 1A	**'F'** 19	**'E'** 18
20					**61** 23	**62** 22	**63** 21	**64** 20

Alternative View of Memory Map



(a) Big-endian (b) Little-endian

Standard...What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Moterola 680x0 (Mac), and most RISC are big-endian
- Internet is big-endian
  - —Makes writing Internet programs on PC more awkward!
  - —WinSock provides htoi and itoh (Host to Internet & Internet to Host) functions to convert