WEEK3

Chapter 10

Instruction Set Architecture Characteristics and Functions

Instruction Set Architecture (ISA)

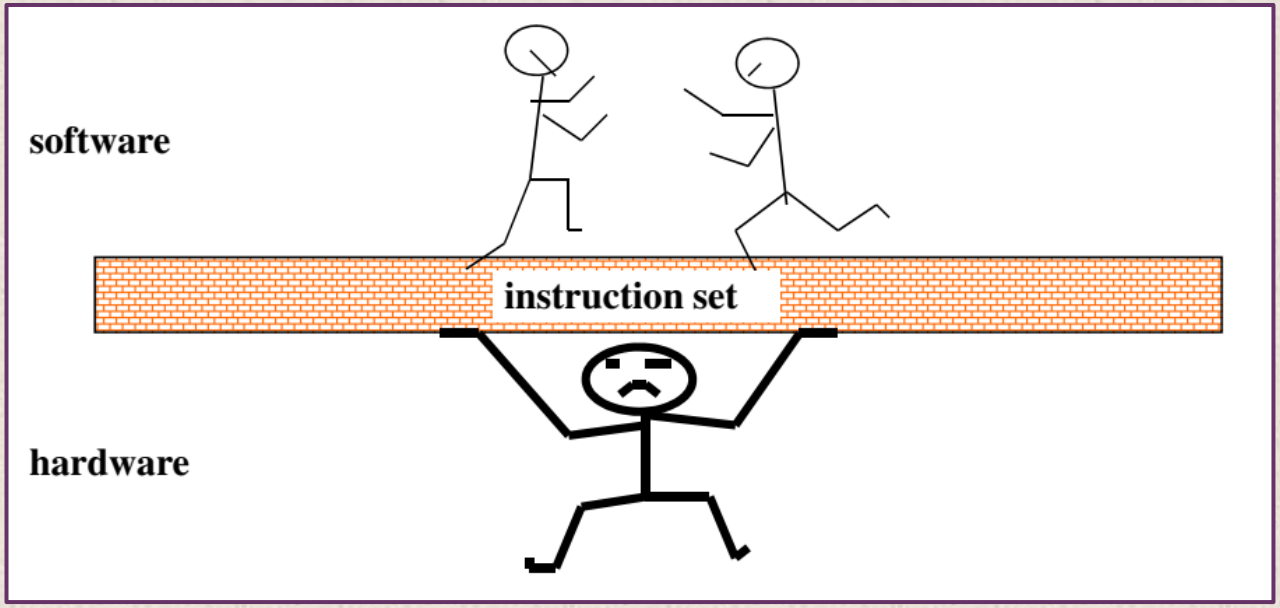
- Complete set of instructions used by a machine
- Abstract interface between the HW and lowest level SW.
- An ISA includes the following …
  - Instructions and Instruction Formats
    - Data Types, Encodings, and Representations
    - Programmable Storage: Registers and Memory
    - Addressing Modes: 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

Instruction Set Architecture (ISA)

- 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 (ISA)



- 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

	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		

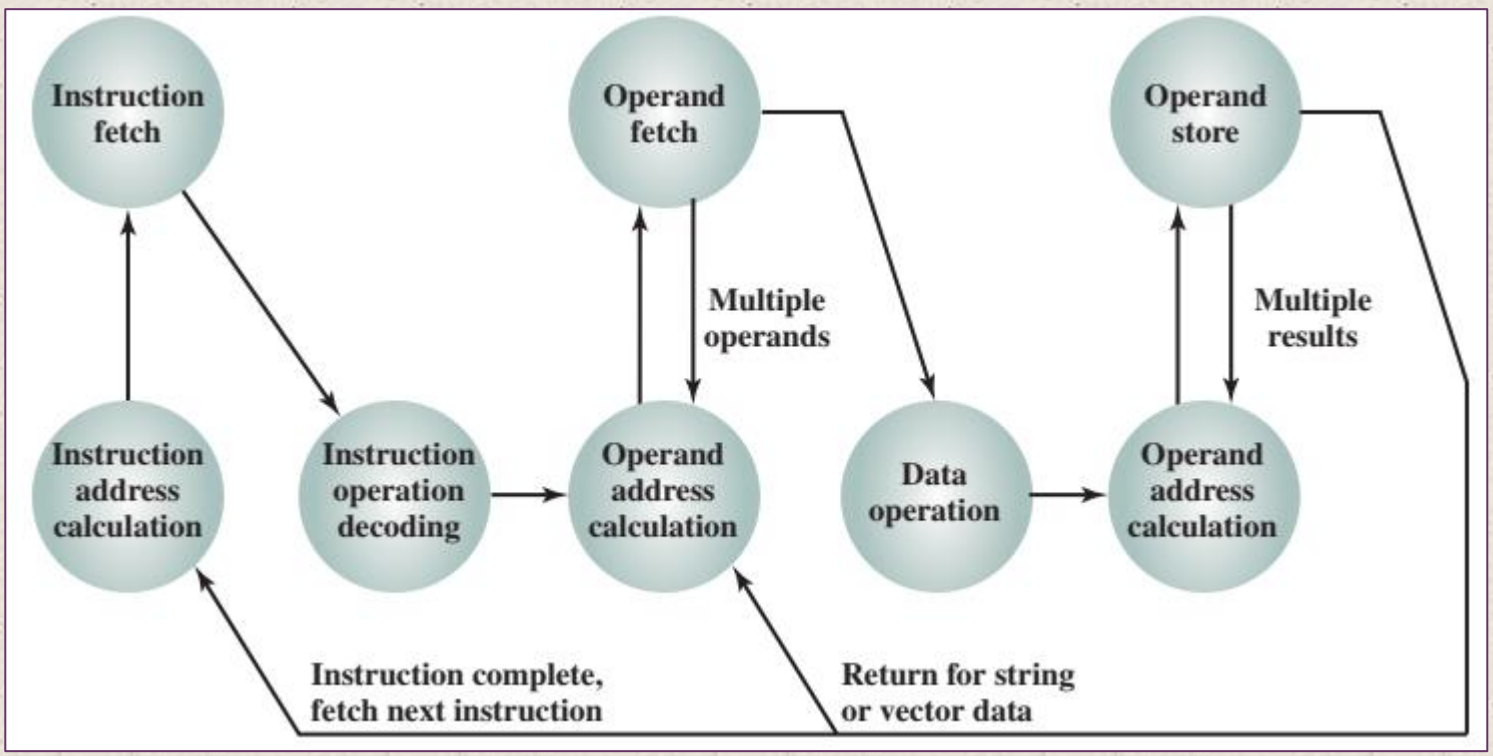
Elements of an Instruction

- 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

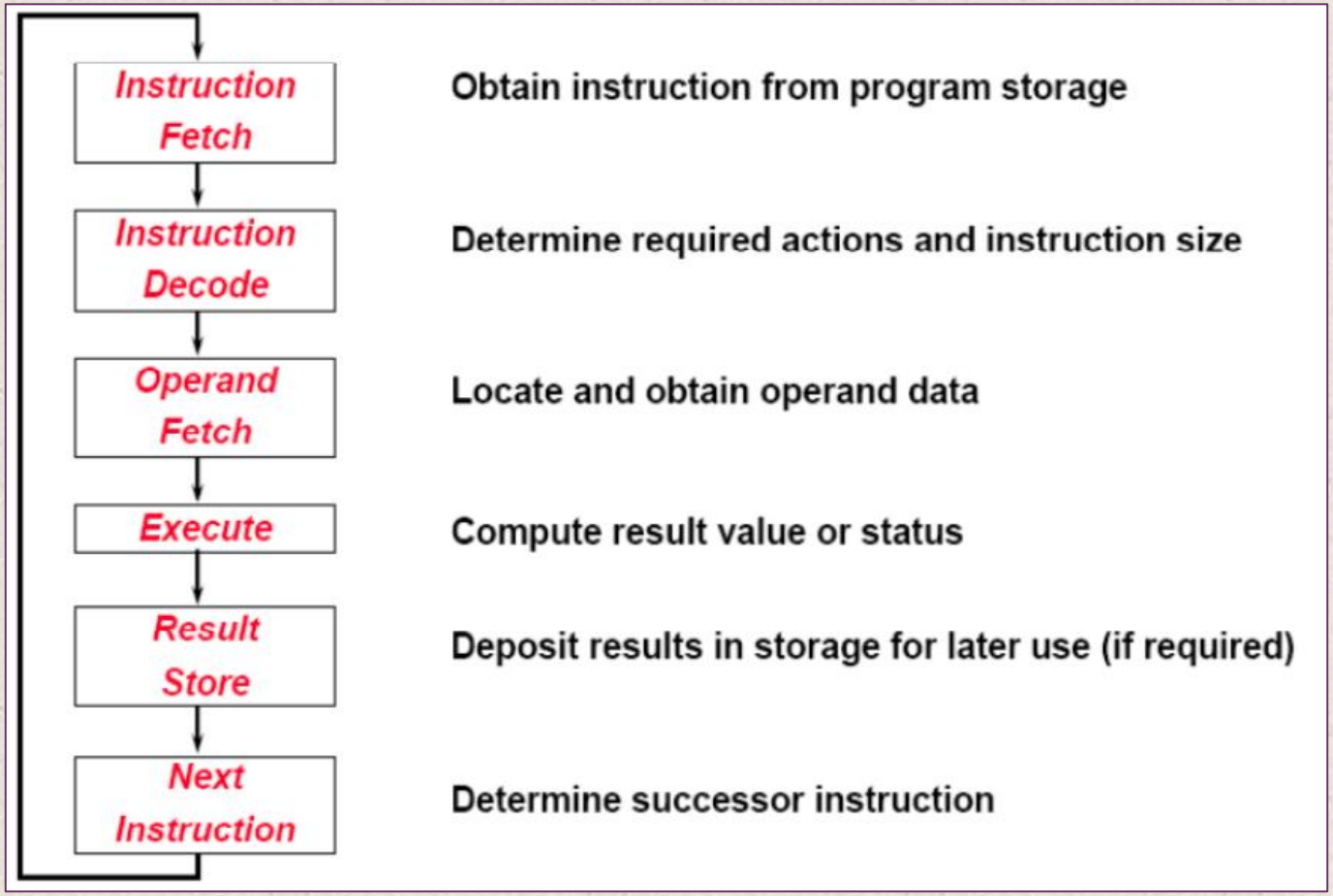
Instruction Representation

- 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



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

Types of Operand

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

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 Operations

- 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

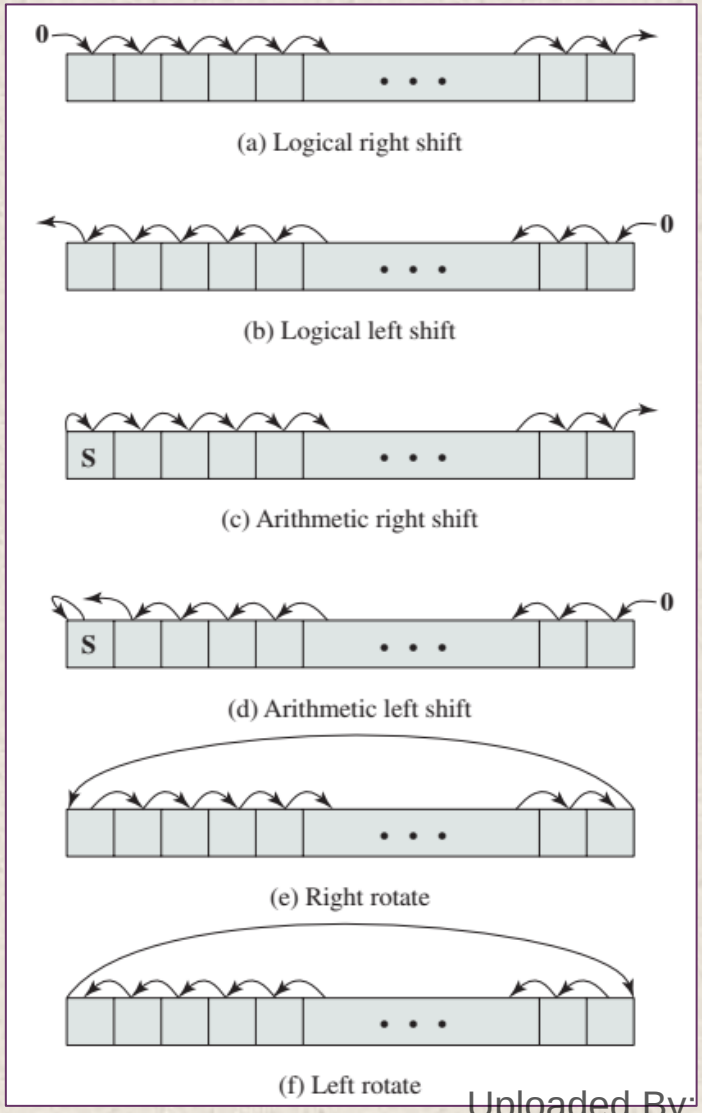
- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Shift & Rotate

- Shift and Rotate Operations

Table 12.7 Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101



Logical & Conversion

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

Input/Output & System Control

- 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

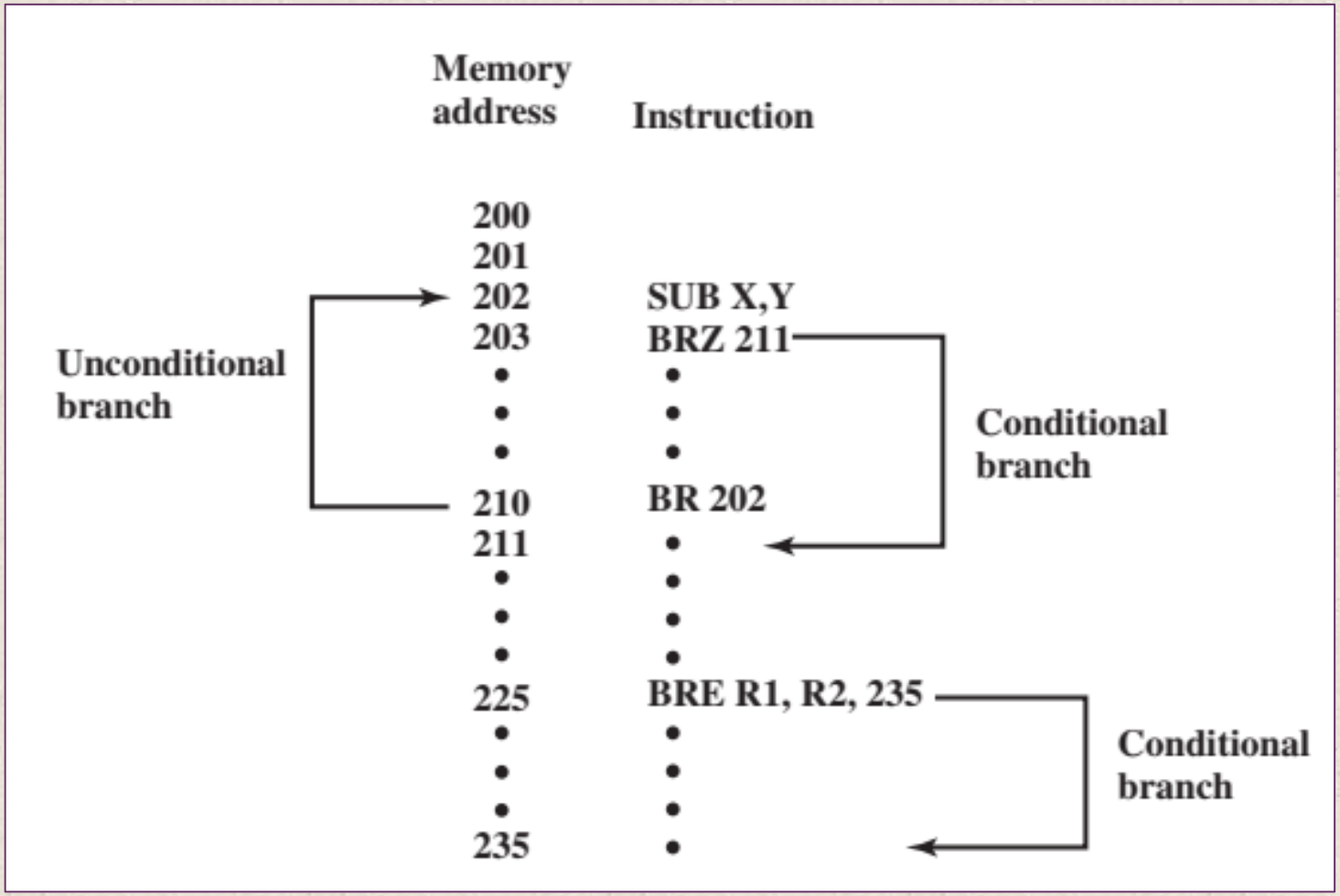
- Branch
  - e.g. ***BRZ X*** branch to x if result of (ADD,SUB,…) is zero
  - See next slide
- Skip
  - e.g. increment and skip if zero ISZ

```
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



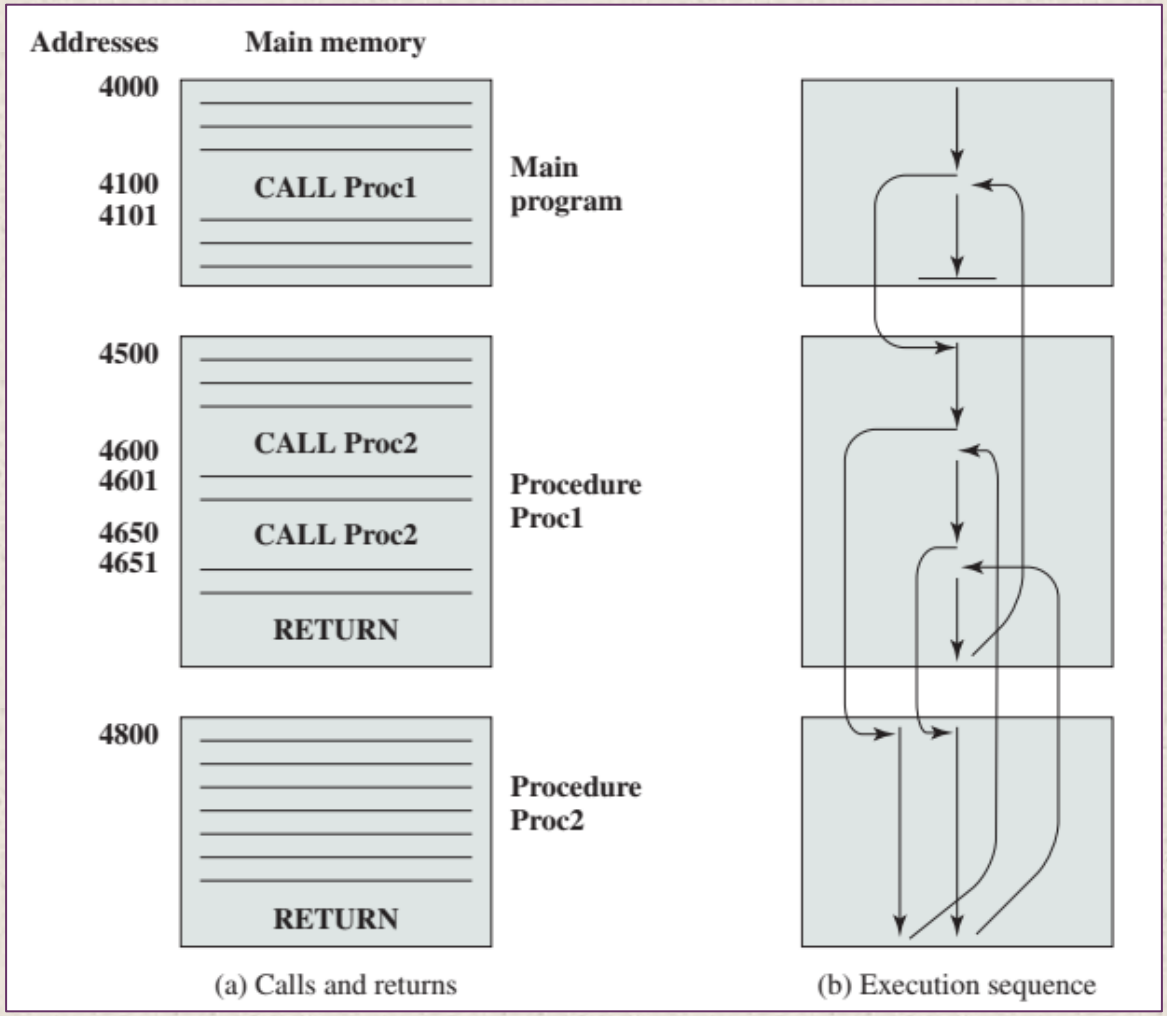
Procedure Calls Instructions

- 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 **time 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

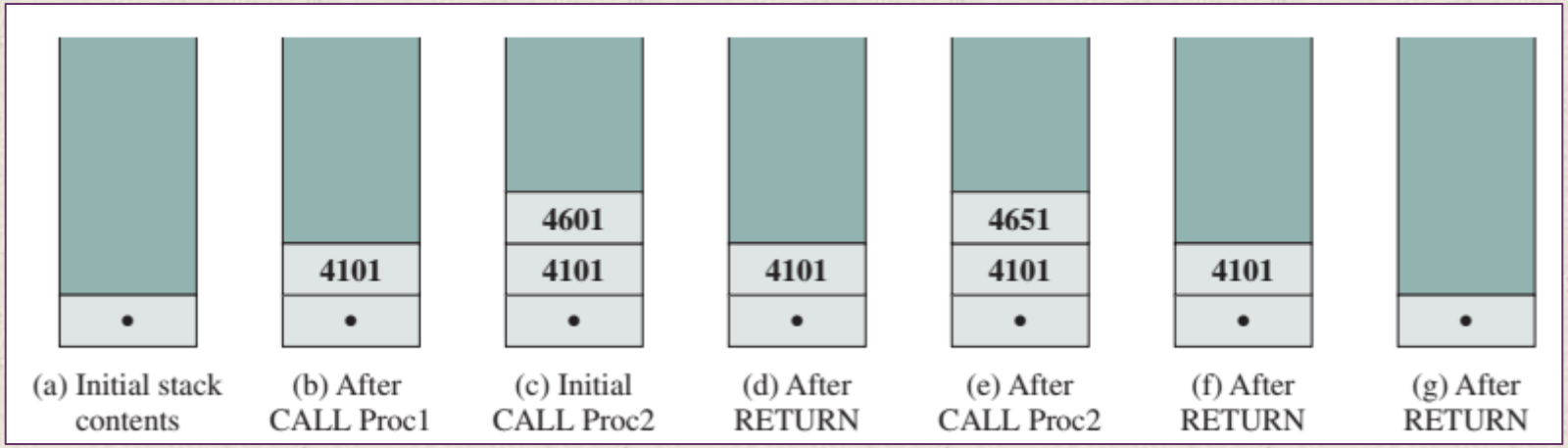


Table 12.1
Utilization of Instruction Addresses (Nonbranching Instructions)

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ¬ B OP C
2	OP A, B	A ¬ A OP B
1	OP A	AC ¬ AC OP A
0	OP	T ¬ (T – 1) OP T

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

Number of Addresses

- # 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

- 2 addresses
  - One address doubles as operand and result
  - ADD a,c (a = a + c)
  - 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

Number of Addresses

- 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

Instruction		Comment
SUB	Y, A, B	Y ¬ A – B
MPY	T, D, E	T ¬ D ´ E
ADD	T, T, C	T ¬ T + C
DIV	Y, Y, T	Y ¬ Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ¬ A
SUB	Y, B	Y ¬ Y – B
MOVE	T, D	T ¬ D
MPY	T, E	T ¬ T ´ E
ADD	T, C	T ¬ T + C
DIV	Y, T	Y ¬ Y ÷ T

(b) Two-address instructions

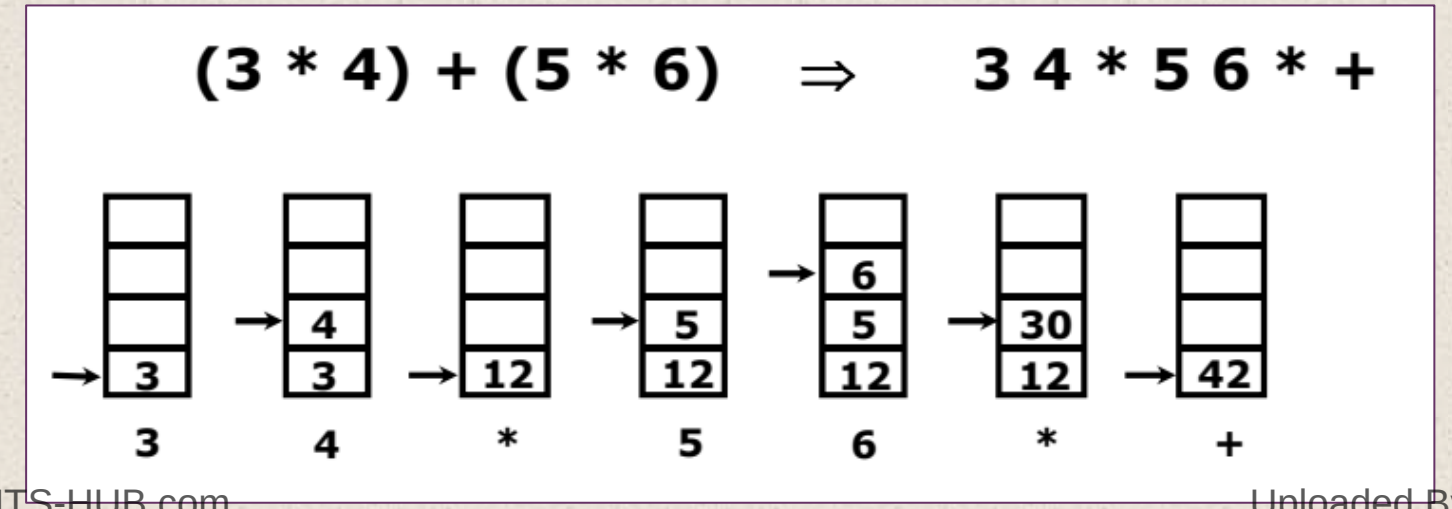
Instruction		Comment
LOAD	D	AC ¬ D
MPY	E	AC ¬ AC ´ E
ADD	C	AC ¬ AC + C
STOR	Y	Y ¬ AC
LOAD	A	AC ¬ A
SUB	B	AC ¬ AC – B
DIV	Y	AC ¬ AC ÷ Y
STOR	Y	Y ¬ AC

(c) One-address instructions

Figure 12.3 Programs to Execute $Y = \frac{A - B}{C + (D \times E)}$

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



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

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

Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
187	78	12

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** [Motorola]
- The system on the right has the least significant byte in the highest address
- This is called **little-endian** [Intel]

Example of C Data Structure

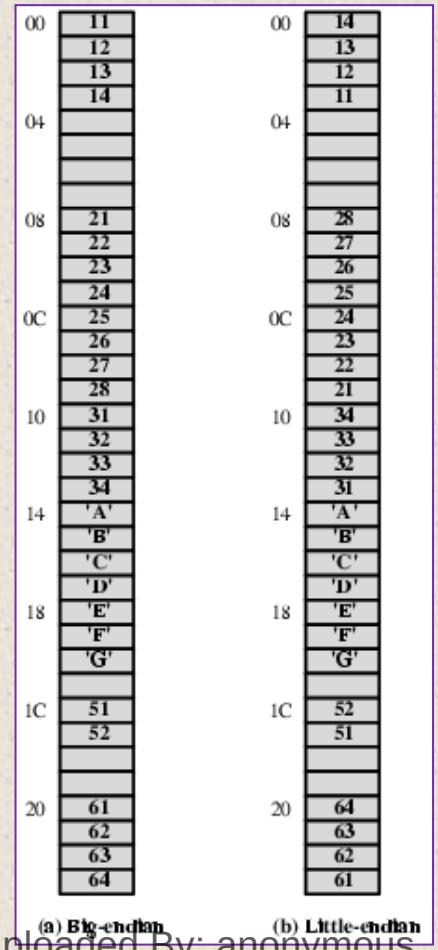
```
Struct{
 int      a;      //0x1112_1314     word
 double     b;     //0x2122_2324_2526_2728     double word
 char  *c;     //0x3132_3334     Word
 char     d[7];     // 'A' 'B' 'C' 'D' 'E' 'F'    'G'      byte array
 short     e;     // 0x5152
 int     f;     //0x6162_6364
}S;
```

Big-endian address mapping

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

Little-endian address mapping

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



Standard…What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Motorola 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