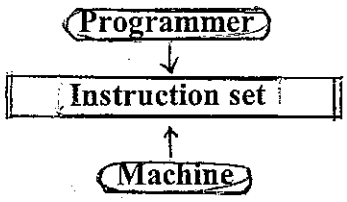
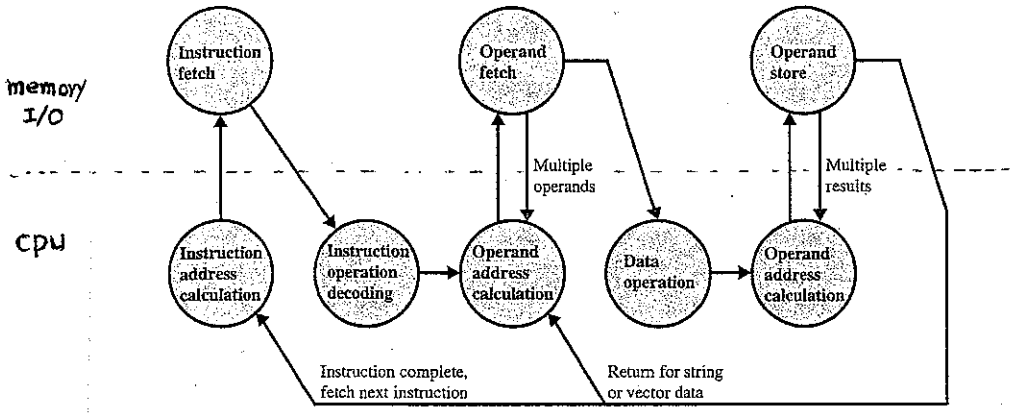
10. Instruction Sets: Characteristics and Functions



register, memory structure

functional requirement for CPU

Instruction cycle



Instruction format

op code	r1	r2
op code	r	m

Op code (symbolic representation: mnemonics)

	Machine 1	Machine 2	Machine 3
Load	LOAD	L	:
Store	STOR	ST	:
Add	ADD	A	:
Subtract	SUB	S	:
Multiply	MUL	M	:
Divide	DIV	D	:

Instruction type

Data processing	-	arithmetic, logical
Data storage	-	load, store
Data movement	-	read, write
Control	-	test, branch

Number of addresses

3-address instruction	op A,B,C	A ← B op C
2-address instruction	op A,B	A ← A op B
1-address instruction	op A	AC ← AC op A
0-address instruction	op	T ← (T-1) op T

Note. Stack computer: Burroughs B5000
Most computer systems use 2 or 3 address instructions.

Instruction Set Design

Issues:

- Operation repertoire
- Data type - integer, float, decimal, character, logical, ...
- Instruction format: length, #addresses, field size, ...
- Registers – no. of registers, kind (general, floating-point, ...)
- Addressing mode – direct / indirect

$Y = (A - B) / (C + D * E)$ → Y = AB-CDE*+/ (postfix)

(a) 3-addr

```
SUB Y,A,B
MPY T,D,E
ADD T,T,C
DIV Y,Y,T
```

(b) 2-addr

```
MOV Y,A
SUB Y,B
MOV T,D
MPY T,E
ADD T,C
DIV Y,T
```

(c) 1-addr

```
LOAD D
MPY E
ADD C
STOR Y
LOAD A
SUB B
DIV Y
STOR Y
```

(d) 0-addr

```
PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MPY
ADD
DIV
POP Y
```

Types of Operands

- **Address** — unsigned integer
- **Numbers**
  - . **integer (fixed-point)**
  - . **real (floating-point)**
  - . **decimal**

0	0000
1	0001
:	:
9	1001

Example.

(1) +12345 (IBM S/370)

unpacked decimal:

F1	F2	F3	F4	C5

packed decimal:

1 2	3 4	5 C

(2) -314

unpacked decimal:

F 3	F 1	D 4

packed decimal:

3 1	4 D

- **characters**
  - . ASCII – 7 bits
  - . EBCDIC – 8 bits
- **logical data**
  - . bit-level operations
  - . Assembly language (some high-level languages)

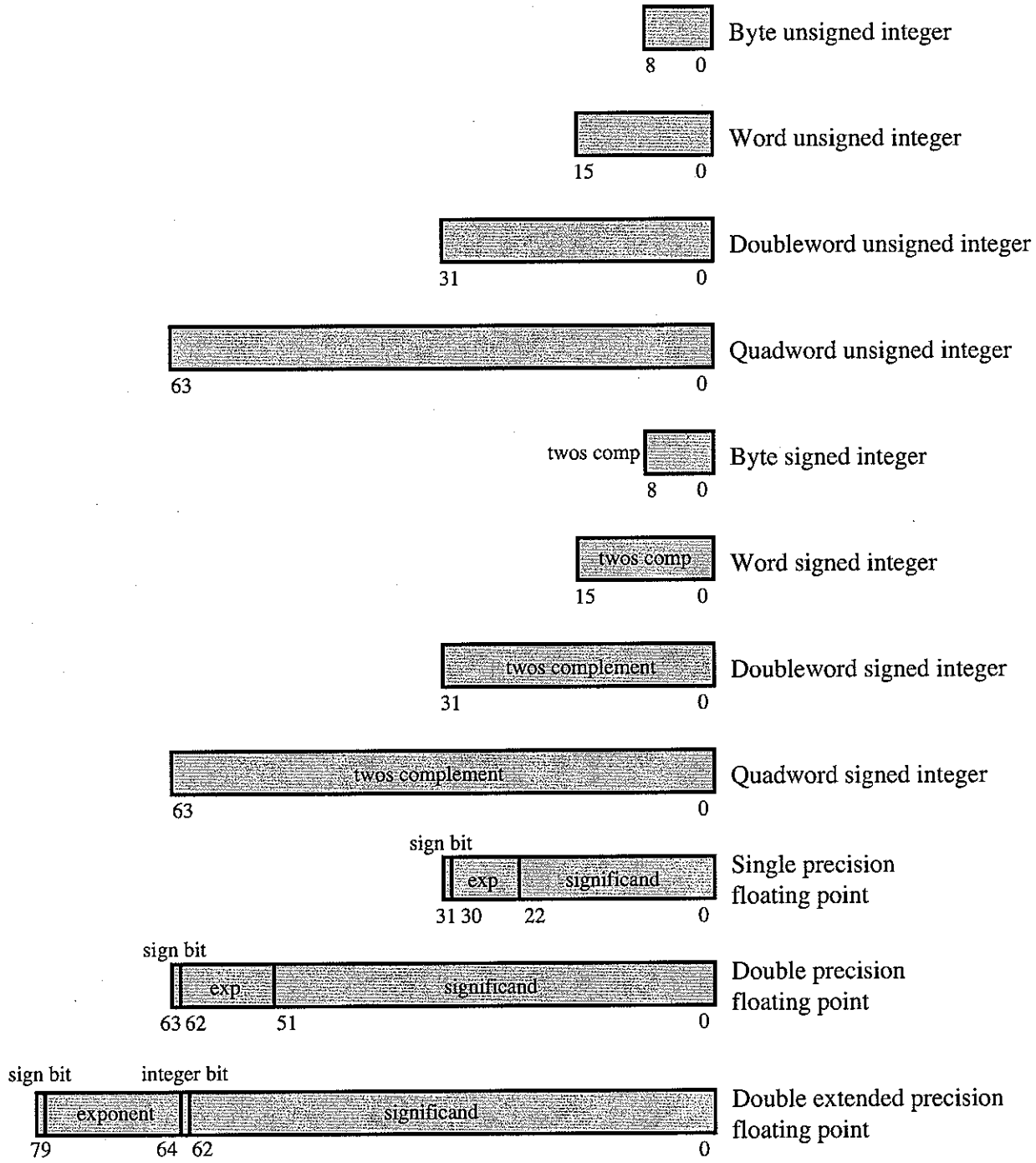


Figure 10.4 Pentium Numeric Data Formats

TABLE 10.3 Common Instruction Set Operations

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	Computer 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 the specified logical operation bitwise
	OR	
	NOT (Complement)	
	Exclusive-OR	
	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 or 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
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
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)

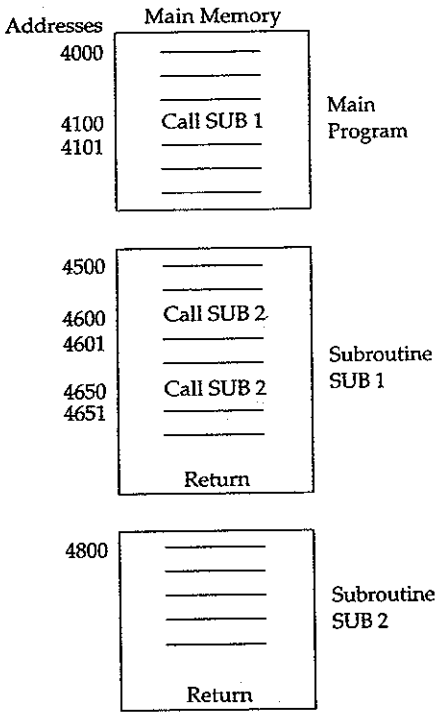
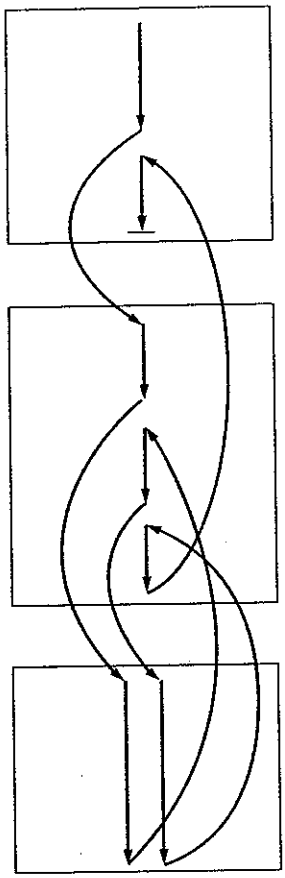


FIGURE 10.7 Nested subroutines



Execution sequence

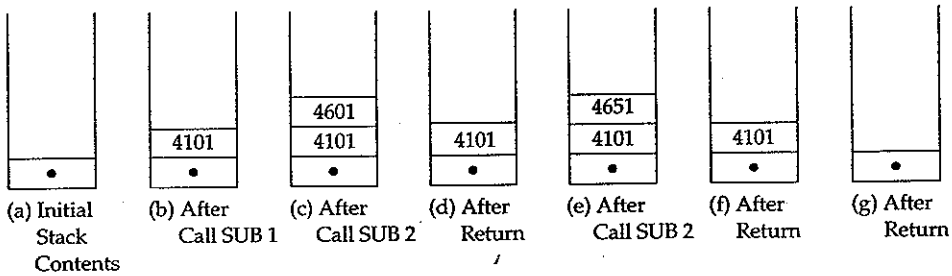


FIGURE 10.8 Use of stack to implement nested subroutines of Figure 10.7

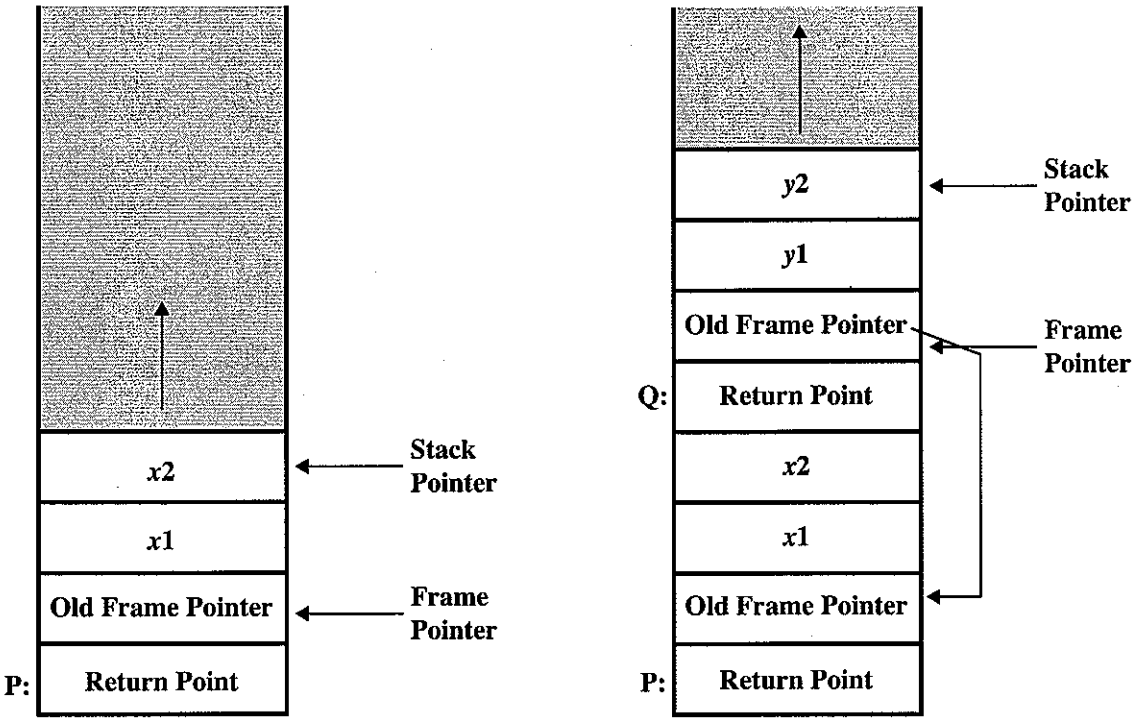


Figure 10.9 Stack Frame Growth Using Sample Procedures P and Q

Assembly Language

Address	Contents
101	0010 0010 0000 0001
102	0001 0010 0000 0010
103	0001 0010 0000 0011
104	0011 0010 0000 0100
201	0000 0000 0000 0010
202	0000 0000 0000 0011
203	0000 0000 0000 0100
204	0000 0000 0000 0000

(a) Binary Program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Hexadecimal Program

```
101  LDA  201
102  ADD  202
103  ADD  203
104  STA  204

201  DAT  2
202  DAT  3
203  DAT  4
204  DAT  0
```

(c) Symbolic Program

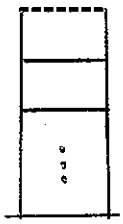
Label	Operation	Opened
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Assembly Program

FIGURE 10.11. Computation of the formula N = I + J + K

10A STACK

Stack (LIFO list) implementation



← top (register)
← top-1 (register)

Usage

(1) sub-procedure call/return
(2) expression evaluation

<stmt> → <assgn stmt> | <cond stmt> | <input stmt> | <while stmt> | ..
<assgn stmt> → <var> := <expr>
<expr> → <expr> + <expr> | <expr> * <expr> | <var>
<var> → a | b | c

Ex. a + b * c (infix)

. precedence: * > +, a + b * c → a + (b * c)

. associativity: left- or right-
a + b + c → (a + b) + c (left-associative)
a ** b ** c → a ** (b ** c) (right-associative)

postfix (reverse Polish):

a b c * +

Advantage:

Expression in postfix can be easily evaluated with a stack.
See Fig 10.16

Note. Many compilers convert infix to postfix, then generate machine code. See Fig 10.17

	Stack	General Registers	Single Register
	Push a Push b Subtract Push c Push d Push e Multiply Add Divide Pop f	Load G[1], a Subtract G[1], b Load G[2], d Multiply G[2], e Add G[2], c Divide G[1], G[2] Store G[1], f	Load d Multiply e Add c Store f Load a Subtract b Divide f Store f
Number of Instructions	10	7	8
Memory Access	10 op + 6 d	7 op + 6 d	8 op + 8 d

Figure 10.15 Comparison of Three Programs to Calculate f = (a − b)/(c + d × e).

Postfix : a b − c d e * + /

Algorithm

Scan postfix expression from left to right.

1. If the element is a variable or constant, push it onto the stack.
2. If the element is an operator, pop the top two items of the stack, perform the operation, and push the result.

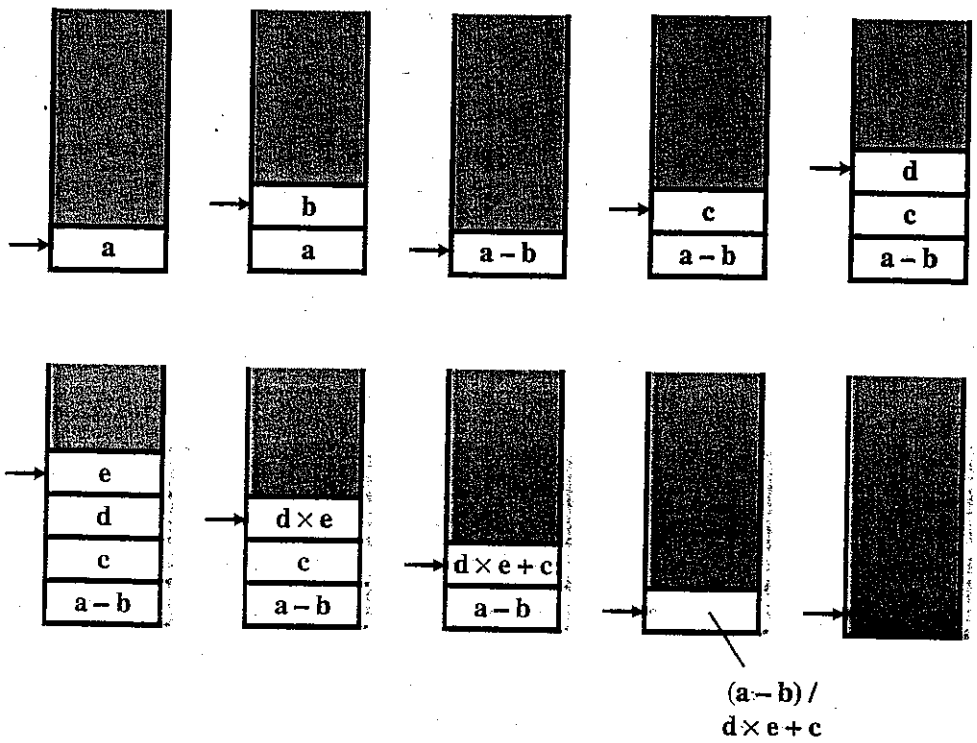


Figure 10.16 Use of Stack to Compute f = (a − b)/(d × e + c).

Input	Output	Stack (top on right)
A + B × C + (D + E) × F	empty	empty
+ B × C + (D + E) × F	A	empty
B × C + (D + E) × F	A	+
× C + (D + E) × F	A B	+
C + (D + E) × F	A B	+ ×
+ (D + E) × F	A B C	+ ×
(D + E) × F	A B C × +	+
D + E) × F	A B C × +	+ (
+ E) × F	A B C × + D	+ (
E) × F	A B C × + D	+ (+
) × F	A B C × + D E	+ (+
× F	A B C × + D E +	+
F	A B C × + D E +	+ ×
empty	A B C × + D E + F	+ ×
empty	A B C × + D E + F × +	empty

Figure 10.17 Conversion of an Expression from Infix to Postfix Notation.

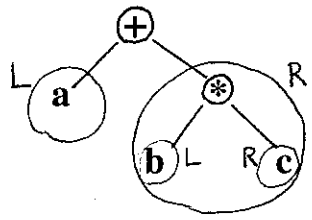
*	+ (D + E) × F	A B C	+
	+ (D + E) × F	A B C × +	empty

[DIJK63]. The infix expression is scanned from left to right, and the postfix expression is developed and output during the scan. The steps are as follows:

1. Examine the next element in the input.
2. If it is an operand, output it.
3. If it is an opening parenthesis, push it onto the stack.
4. If it is an operator, then
   - If the top of the stack is an opening parenthesis, then push the operator.
   - If it has higher priority than the top of the stack (multiply and divide have higher priority than add and subtract), then push the operator.
   - Else, pop operation from stack to output, and repeat step 4.
5. If it is a closing parenthesis, pop operators to the output until an opening parenthesis is encountered. Pop and discard the opening parenthesis.
6. If there is more input, go to step 1.
7. If there is no more input, unstack the remaining operands.

Figure 10.17 illustrates the use of this algorithm. This example should give the reader some feel for the power of stack-based algorithms.

Binary Tree Traversal



(rt) L R → 3! different ordering

(rt)	L	R	≡ preorder	(+ a * b c)
L	(rt)	R	≡ inorder	(a + b * c)
L	R	(rt)	≡ postorder	(a b c * +)

Note. Postorder
- Reverse Polish notation
- Jan Łukasiewicz
- Parenthesis free

Note.
- inorder + preorder → unique binary tree
- inorder + postorder → unique binary tree

Infix → Postfix
- using stack (Fig 10.17)
- using parenthesis

Little-Endian, Big-Endian

Ex. 12 34 56 78h

184	12	184	78
185	34	185	56
186	56	186	34
187	78	187	12
big-endian		little-endian	
IBM S/370		Intel	
Motorola		Vax	
Sun SPARC		Alpha	