William Stallings
Computer Organization
and Architecture
7th Edition

Chapter 10

Instruction Sets:

Characteristics and Functions

Key points

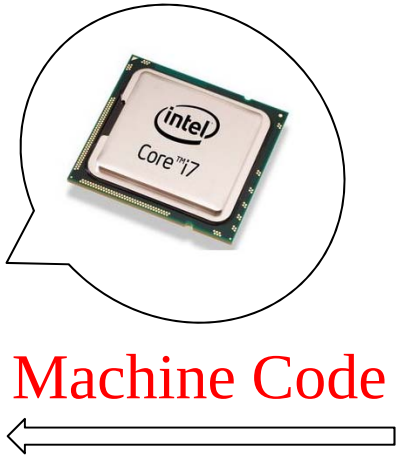
- Opcodes specify operations to be performed
- Categories of operations: arithmetic and logic operations, movement, I/O, control
- Types of data: address, number, character, logical data
- Stacks are used to manage processing calls and returns
- Processors may categorized as big-endian, little-endian, or bi-endian

10.1 machine instruction characteristics

- Elements of a machine instruction
- Instruction representation
- Instruction type
- Number of addresses
- Instruction set design

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes



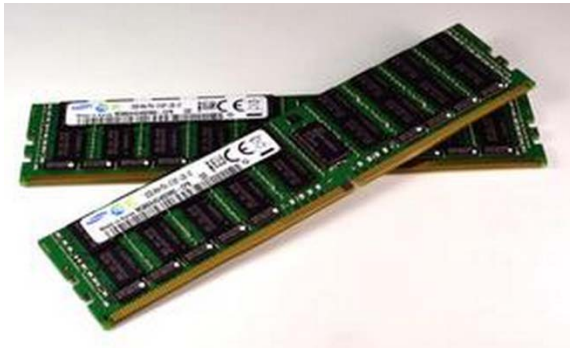
Machine Code

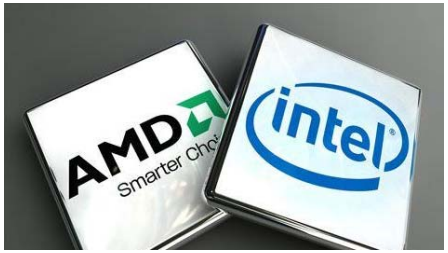
Elements of an Instruction

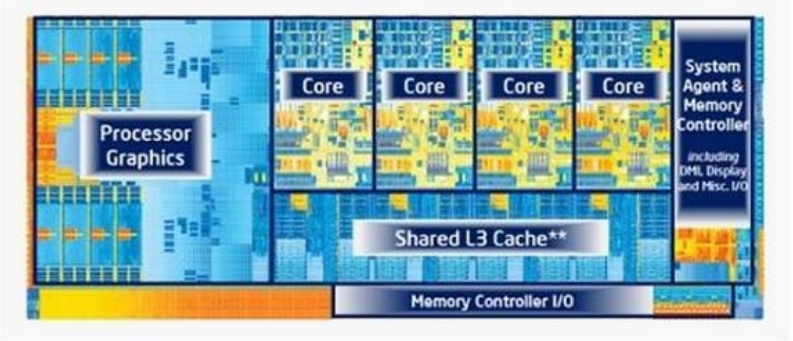
- Operation code (Op code)
  - Operations to be performed (binary code)
  - Do this
- Source Operand reference
  - Operands that are inputs for the operation
  - To this
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - Where to fetch the next instruction
  - When you have done that, do this...

Where have all the Operands Gone?

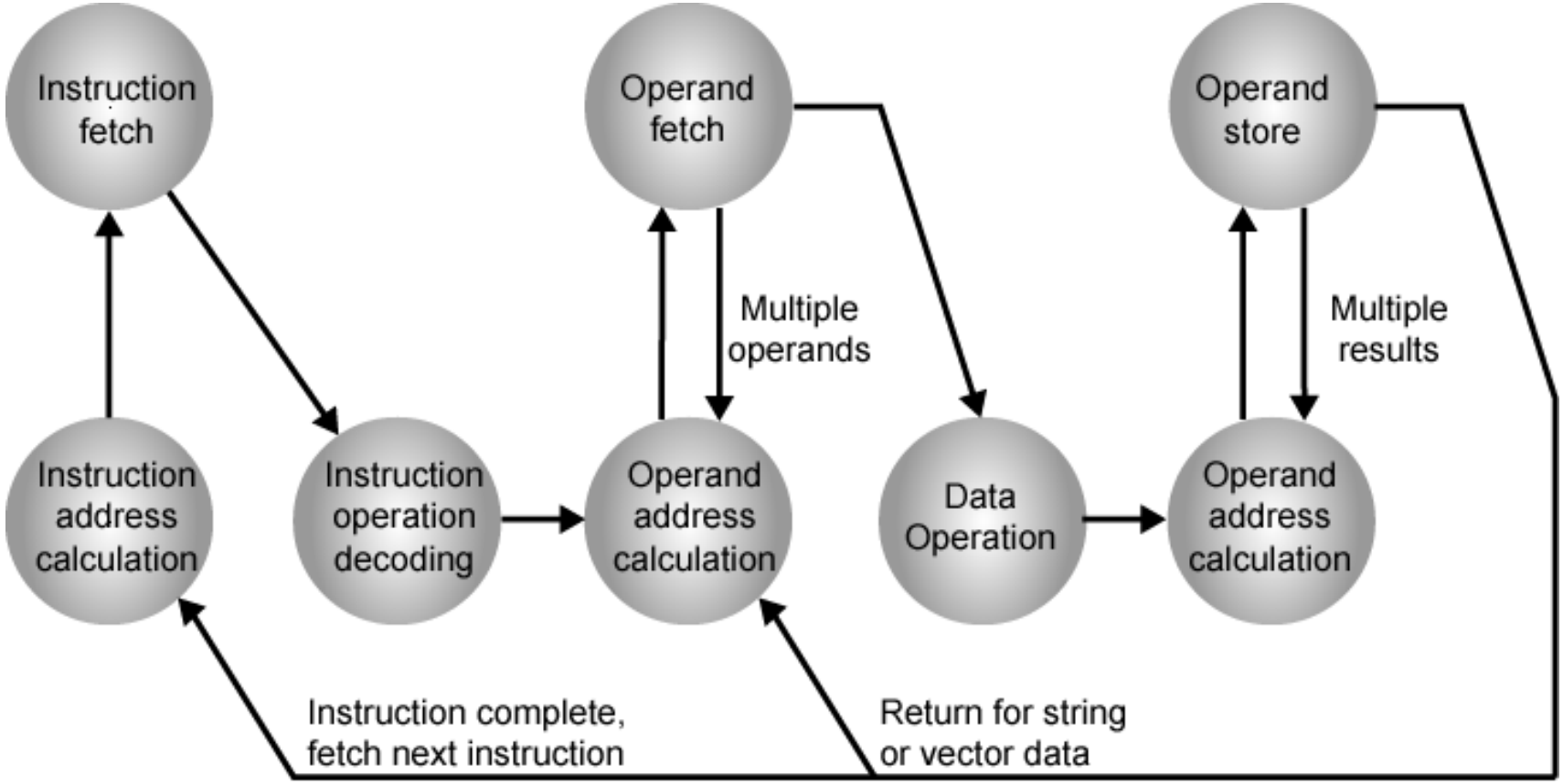
- Main memory (or virtual memory or cache)
- CPU register
- I/O device







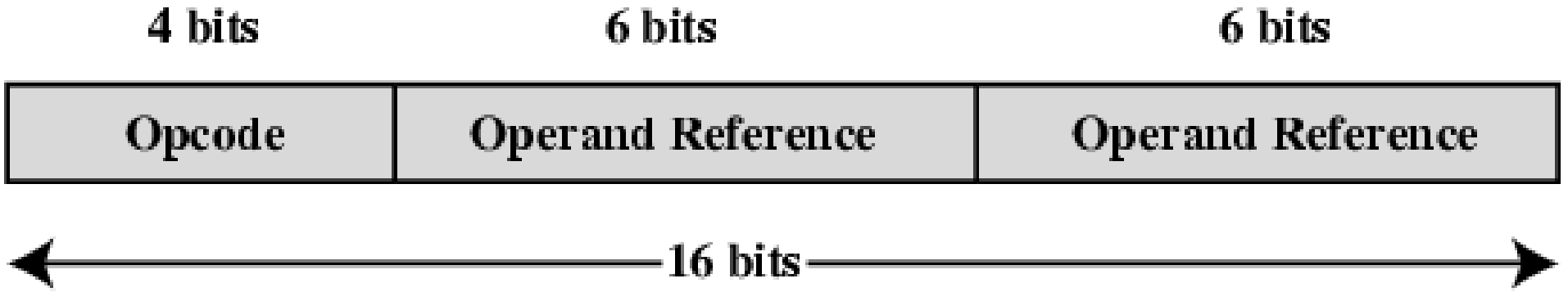
Instruction Cycle State Diagram



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

Simple Instruction Format



From high-level language to machine language

- High level language: eg, x=x+y
- Machine language
  - Load a register with the contents of memory location X;
  - Add the contents of memory location Y to the register;
  - Store the contents of the register in memory location X.

Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Control

Number of Addresses

- Number of addresses needed in an instruction
- 4 addresses
- 3 addresses
- 2 addresses
- 1 addresses
- 0 addresses

Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

Number of Addresses (c)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c

  - c = a + b

How Many Addresses

- More addresses
  - More complex Instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex instructions
  - More instructions per program
  - Faster fetch/execution of instructions

Design Decisions

- Operation repertoire
  - How many and which operations to provide?
  - How complex are they?
- Data types
- Instruction formats
  - Instruction Length
  - Number of addresses
  - Size of various fields
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes

10.2 Types of Operands

- Addresses
- Numbers
  - Integer or fixed point
  - floating point
  - Decimal
- Characters
  - IRA, ASCII etc.
- Logical Data
  - Bits or flags

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

10.4 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 390 (table 10.5)
- Or one instruction and different addresses
  - e.g. VAX

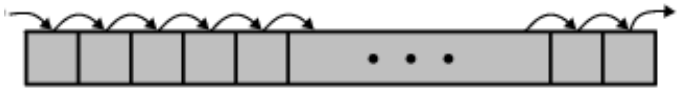
Arithmetic

- Add, Subtract, Multiply, Divide
- For signed Integer (fixed point), Floating point, or packed decimal number
- May include
  - Absolute
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)
- Involve data transfer

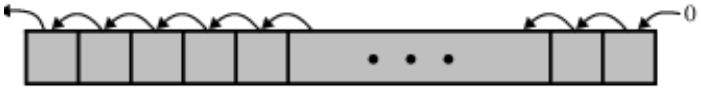
Logical

- Bitwise operations
- Based on Boolean operations
- AND, OR, NOT, XOR
- Eg.
  - (R1)=10100101
  - (R2)=00001111
  - (R1) AND (R2)=00000101
  - (R1) OR (R2) =10101111
  - NOT (R1) =01011010
  - (R1) XOR (R2)=10101010

Shift and Rotate Operations



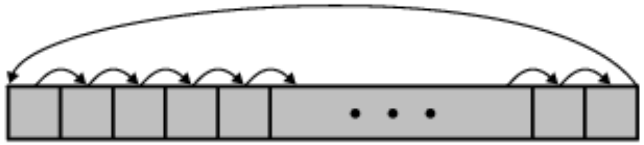
(a) Logical right shift



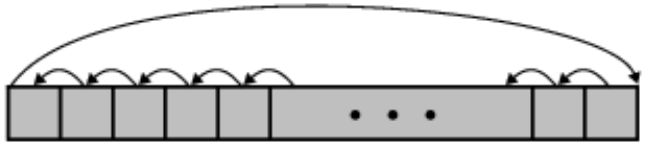
(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

Shifts correspond to multiplication and division

- With number in twos complement notation
- A right arithmetic shift correspond to a division by 2 with truncation for odd numbers
- A left shift correspond to a multiplication by 2 when there is no overflow
- If overflow occurs
  - Logical shift changes the sign
  - Arithmetic shift retains the sign

Conversion

- E.g. Binary to Decimal
- E.g. EBCDIC code to IRA code
  - TR R1, R2, L

Input/Output

- May be specific instructions
- May be memory mapped programmed I/O, DMA
- May be use of I/O processor

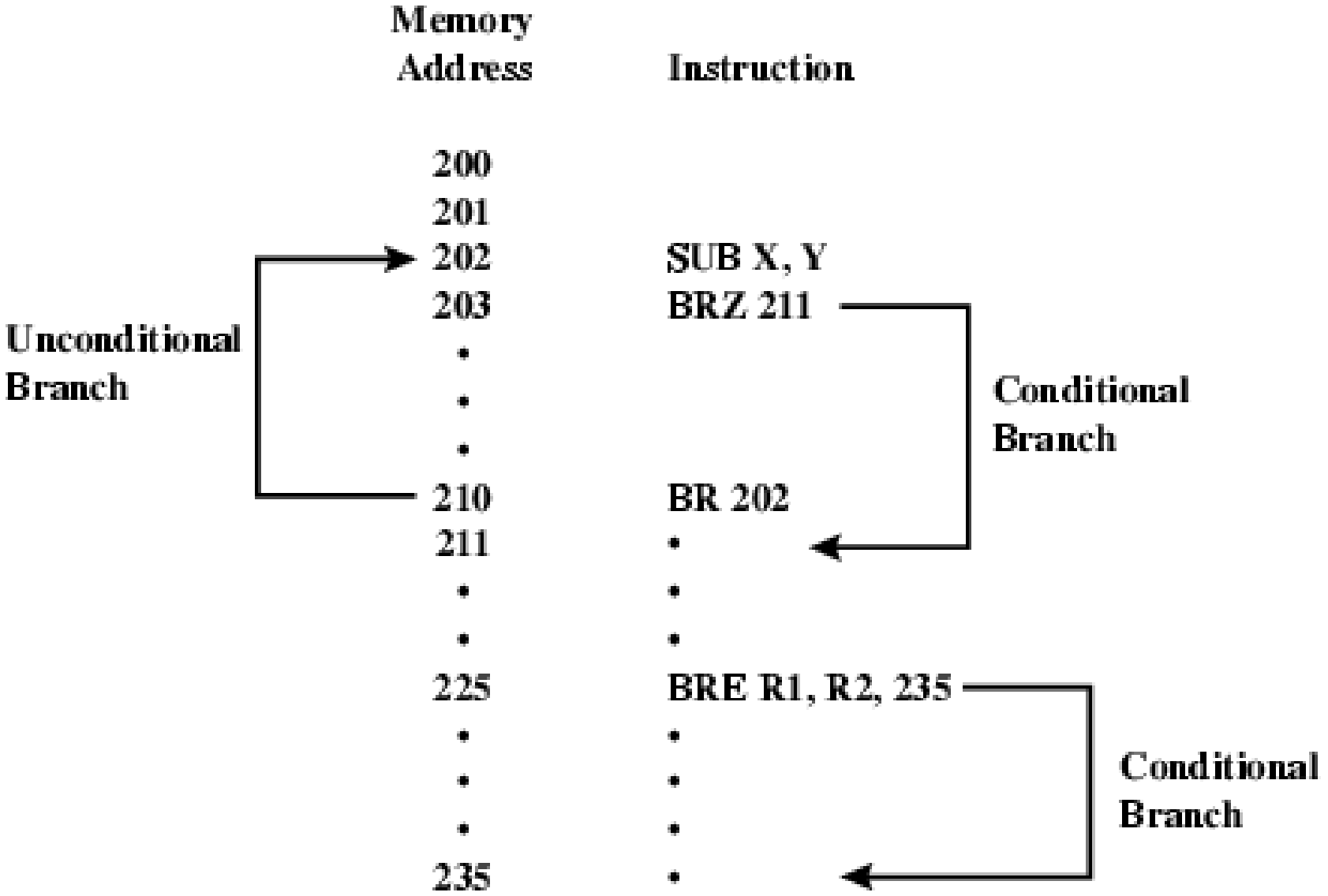
Systems Control

- Privileged state or privileged area of memory
- CPU needs to be in specific state
- Privileged instructions are reserved for operating system

Transfer of Control

- Change the sequence of instruction execution
  - Execute a instruction more than once
  - Conditional execution
  - Break a large program into smaller pieces
- Branch
  - e.g. BRZ X: branch to x if result is zero
- Skip
- Subroutine call
  - c.f. interrupt call

Branch Instruction



Skip

- one instruction is skipped
- e.g. increment and skip if zero

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

Procedure call instruction

- A procedure can be called from more than one location
- A procedure call can appear in a procedure
- Each procedure call is matched by a return in the called program

Return address store

- Register

  RN ← PC+Δ

  PC ← X

- Start of called procedure

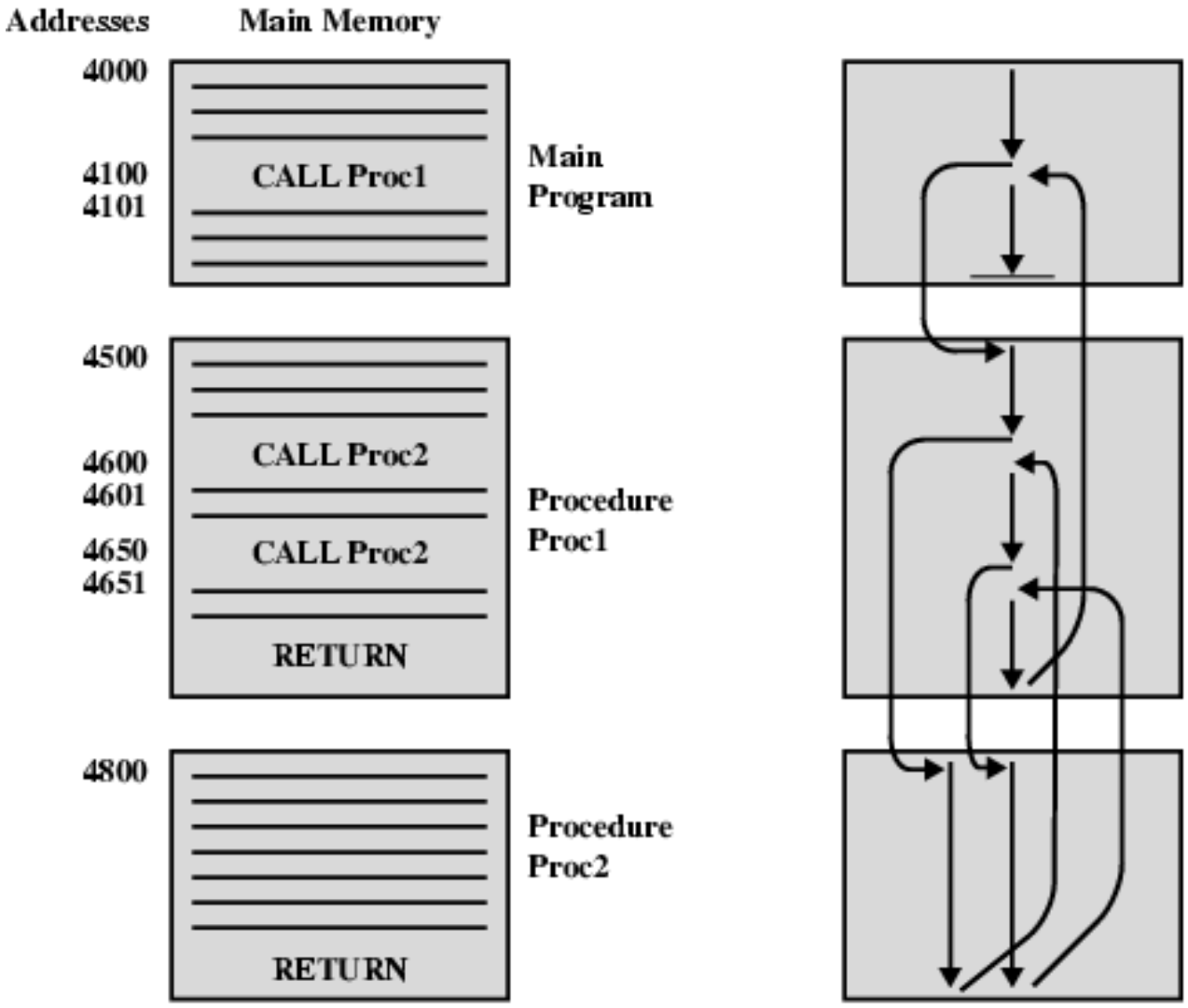
  X ← PC+Δ

  PC ← X+1

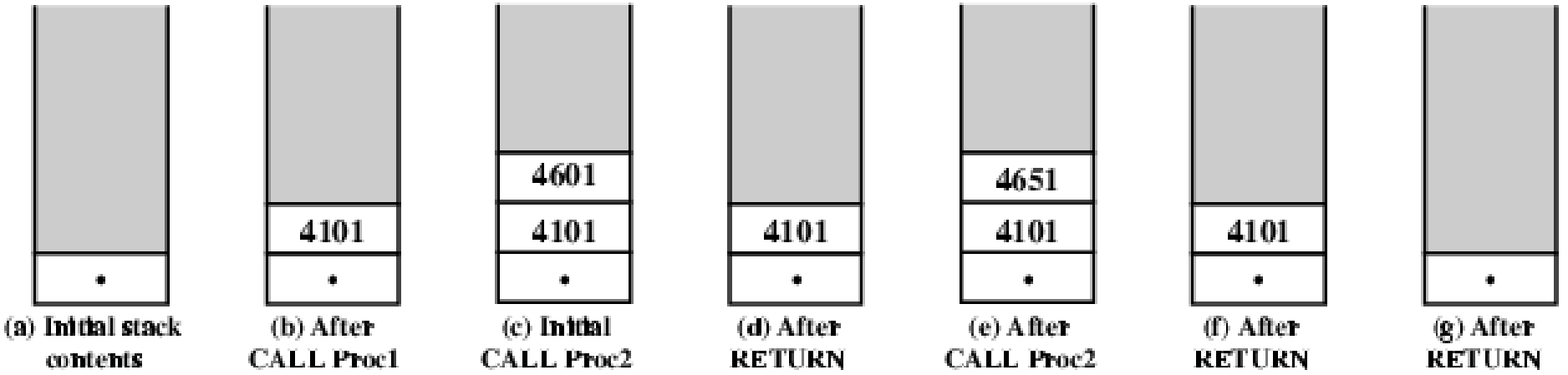
- Top of stack

  PUSH PC+ Δ

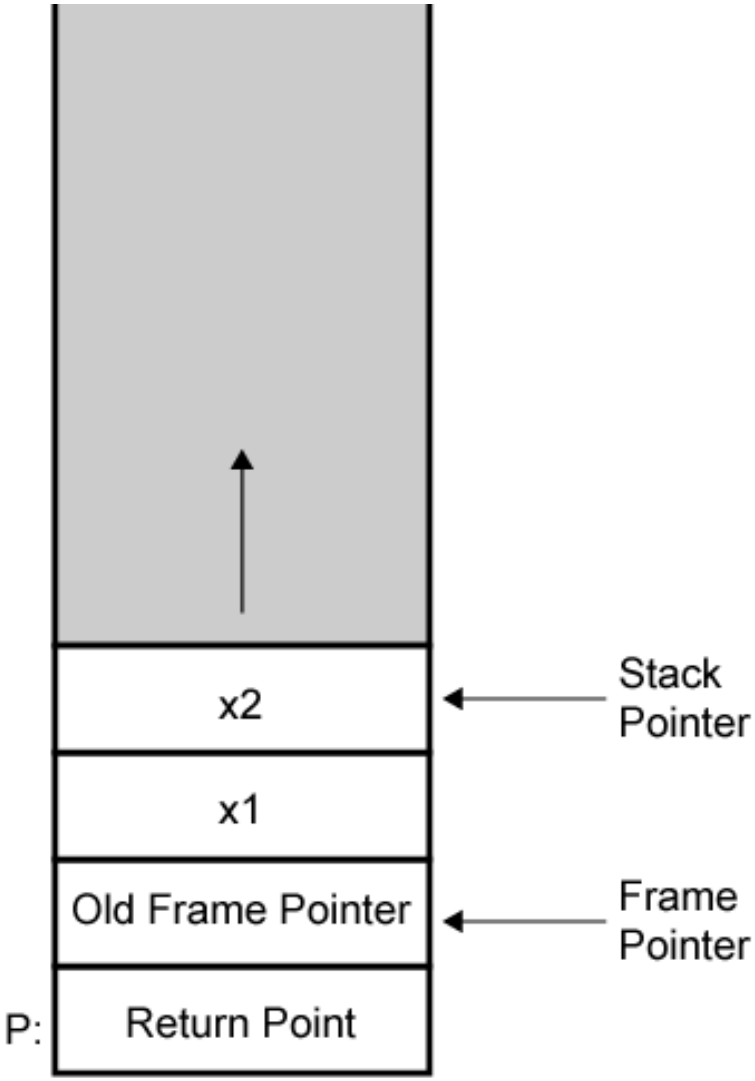
Nested Procedure Calls



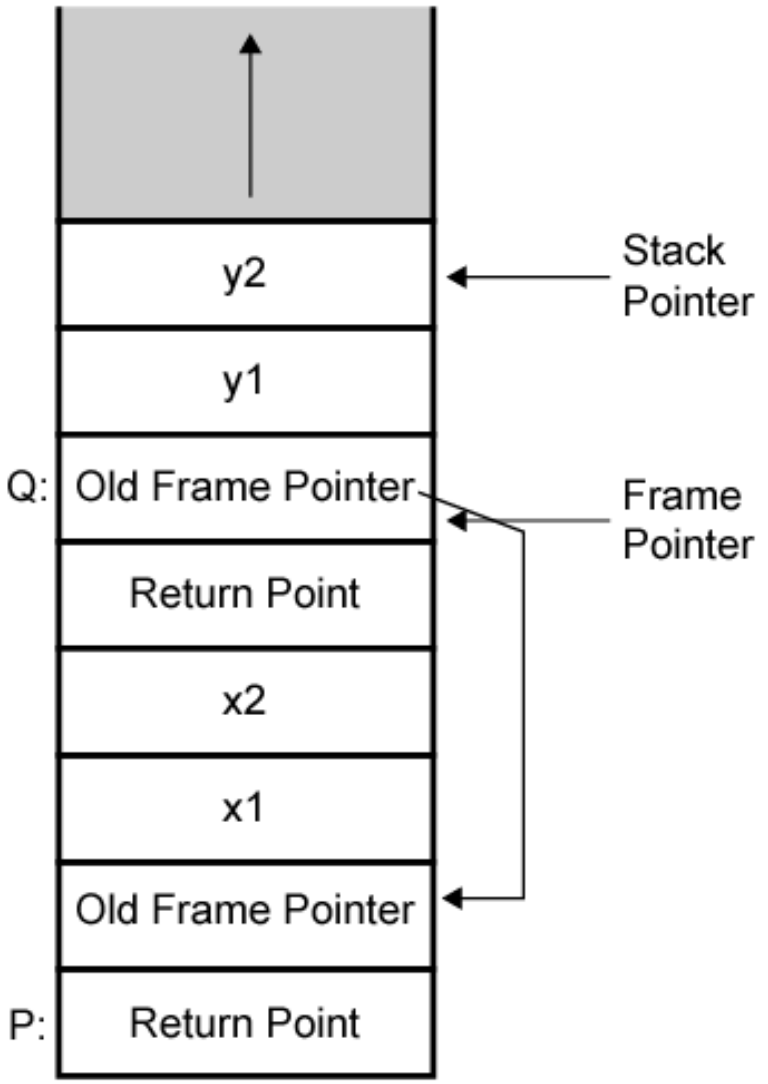
Use of Stack


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

Stack Frame Growth Using Sample Procedures P and Q



(a) P is active



(b) P has called Q

Memory management

- Privileged instructions
- Executed from operating system

Condition codes

- C Carry
- P Parity
- A Auxiliary carry
- Z Zero
- S Sign
- O Overflow

10.6 Assembly language

- Machine instructions are binary numbers
- Write the program in hexadecimal
- Symbolic program
  - —Symbol for opcode
  - —Absolute numeric address
- Assembly language
  - —Symbol for opcode
  - —Symbolic address
  - —Milestone in the evolution of computer technology

Appendix 10A Stacks

- An ordered set of elements
- The point of access is the top of stack
- Last-in-first-out (LIFO) list
  - PUSH: appends one new item to the top of the stack
  - POP: removes the top item from the stack
  - Top of the stack moves
- A contiguous block of location is reserved in main memory for the stack
  - Stack pointer
  - Stack base (high-address end)
  - Stack limit (low-address end)

Appendix 10B Little-, Big- and Bi-endian

- What order do we read numbers that occupy more than one byte
- Big-endian: The system on the left has the least significant byte in the lowest address
- Little-endian: The system on the right has the least significant byte in the highest address

Byte Order (example)

- 12345678 in hex 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

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

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

Homework

- Reading book
- 1.Key Terms
- 2.Problems

(7th Edition)

  - 10.6
  - 10.18
  - 10.19

(6th Edition)

  - 10.3
  - 10.10
  - 10.11