(Advanced) Computer Architechture

Prof. Dr. Hasan Hüseyin BALIK

(5th Week)

Outline

3. Instruction sets
   - —Instruction Sets: Characteristics and Functions
   - —Instruction Sets: Addressing Modes and Formats
   - —Assembly Language and Related Topics

3.1 Instruction Sets: Characteristics and Functions

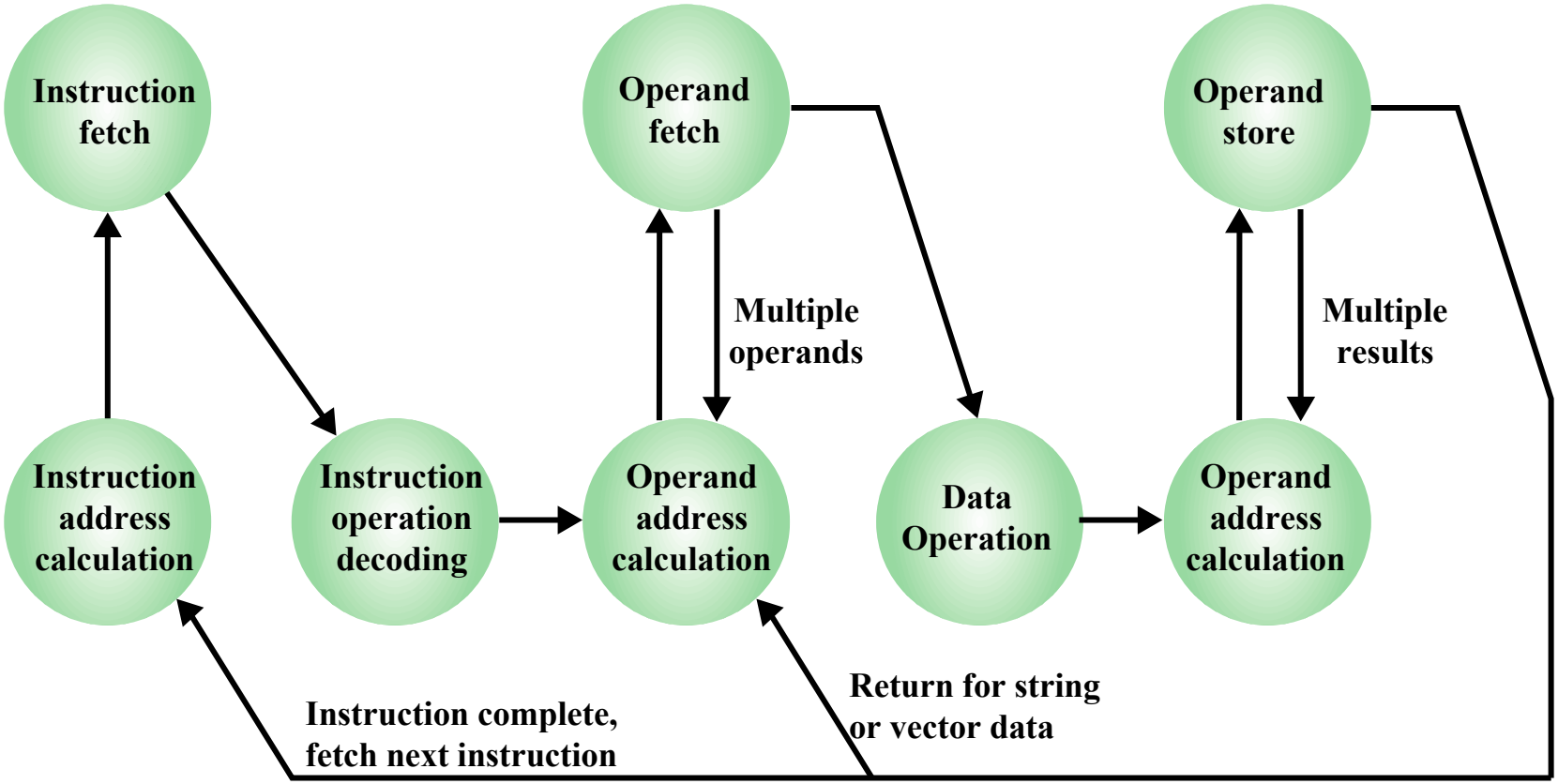
3.1 Outline

- Machine Instruction Characteristics
- Types of Operands
- Intel x86 and ARM Data Types
- Types of Operations
- Intel x86 and ARM Operation Types

Machine Instruction Characteristics

- The operation of the processor is determined by the instructions it executes, referred to as *machine instructions* or *computer instructions*
- The collection of different instructions that the processor can execute is referred to as the processor's *instruction set*
- Each instruction must contain the information required by the processor for execution

Instruction Cycle State Diagram



Machine Instruction Elements:

- Operation code (opcode) : Specifies the operation to be performed
- Source operand reference : operands that are inputs for the operation
- Result operand reference: The operation may produce a result
- Next instruction reference: This tells the processor where to fetch the next instruction

Source and result operands can be in one of four areas:

1) Main or virtual memory
   - As with next instruction references, the main or virtual memory address must be supplied

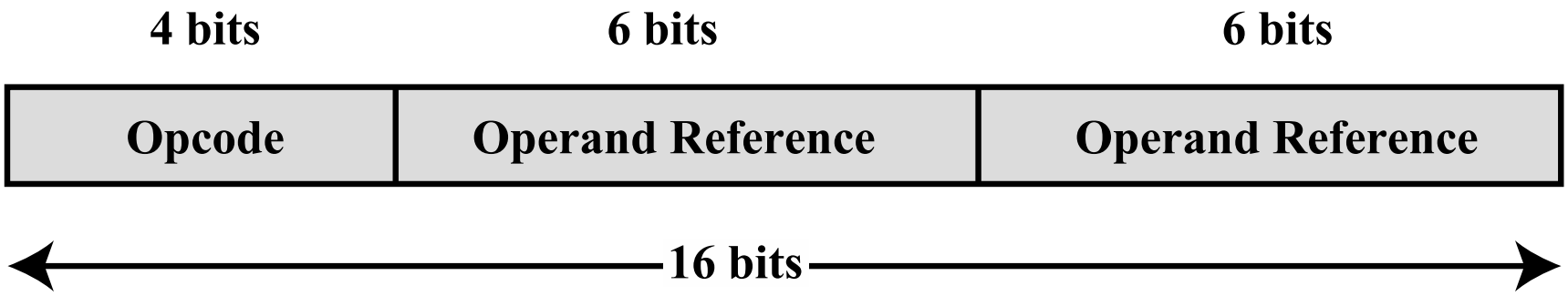
2) I/O device
   - The instruction must specify the I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address

3) Processor register
   - A processor contains one or more registers that may be referenced by machine instructions.
   - If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

4) Immediate
   - The value of the operand is contained in a field in the instruction being executed

A Simple Instruction Format

- Within the computer each instruction is represented by a sequence of bits
- The instruction is divided into fields, corresponding to the constituent elements of the instruction



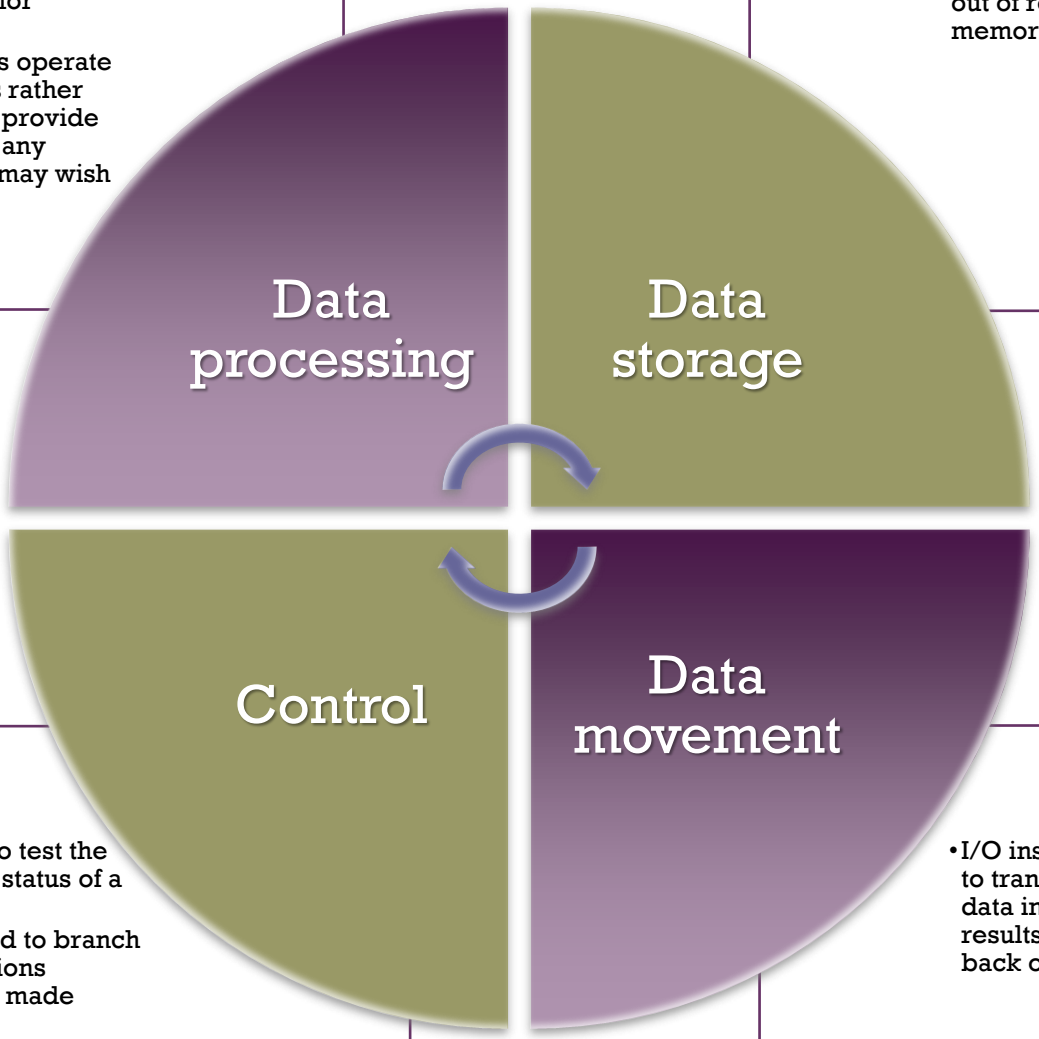
Instruction Representation

- Opcodes are represented by abbreviations called *mnemonics*
- Examples include:
  - ADD Add
  - SUB Subtract
  - MUL Multiply
  - DIV Divide
  - LOAD Load data from memory
  - STOR Store data to memory
- Operands are also represented symbolically (ADD R, Y: may mean add the value contained in data location Y to the contents of register R)
- Each symbolic opcode has a fixed binary representation
  - The programmer specifies the location of each symbolic operand

Instruction Types

- Arithmetic instructions provide computational capabilities for processing numeric data
- Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers, thus they provide capabilities for processing any other type of data the user may wish to employ

- Movement of data into or out of register and or memory locations



- Test instructions are used to test the value of a data word or the status of a computation
- Branch instructions are used to branch to a different set of instructions depending on the decision made

- I/O instructions are needed to transfer programs and data into memory and the results of computations back out to the user

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

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

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

Instruction Set Design

Very complex because it affects so many aspects of the computer system

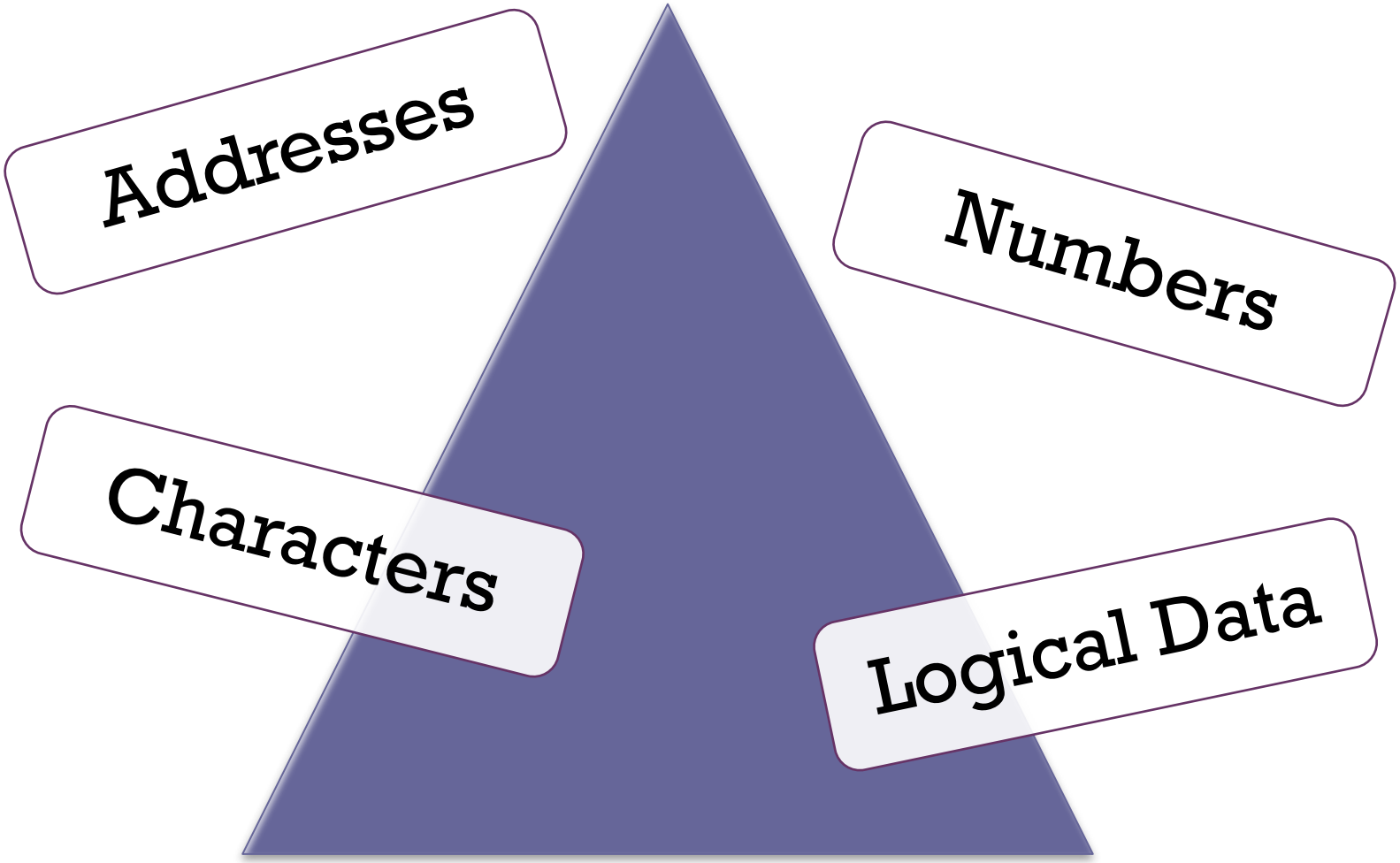
Defines many of the functions performed by the processor

Programmer's means of controlling the processor

Fundamental design issues:

Operation repertoire	Data types	Instruction format	Registers	Addressing
• How many and which operations to provide and how complex operations should be	• The various types of data upon which operations are performed	• Instruction length in bits, number of addresses, size of various fields, etc.	• Number of processor registers that can be referenced by instructions and their use	• The mode or modes by which the address of an operand is specified

Machine instructions operate on



Numbers

- All machine languages include numeric data types
- Numbers stored in a computer are limited:
  - Limit to the magnitude of numbers representable on a machine
  - In the case of floating-point numbers, a limit to their precision
- Three types of numerical data are common in computers:
  - Binary integer or binary fixed point
  - Binary floating point
  - Decimal
- Packed decimal
  - Each decimal digit is represented by a 4-bit code with two digits stored per byte
  - To form numbers 4-bit codes are strung together, usually in multiples of 8 bits

Characters

- A common form of data is text or character strings
- Textual data in character form cannot be easily stored or transmitted by data processing and communications systems because they are designed for binary data
- Most commonly used character code is the International Reference Alphabet (IRA)
  - Referred to in the United States as the American Standard Code for Information Interchange (ASCII)
- Another code used to encode characters is the Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - EBCDIC is used on IBM mainframes

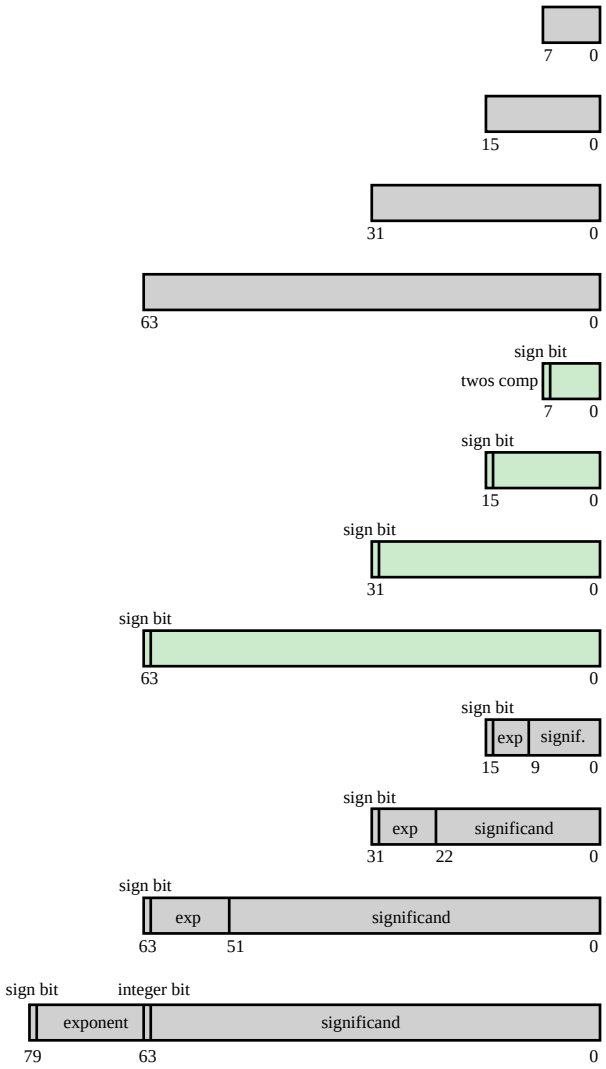
Logical Data

- An *n*-bit unit consisting of *n* 1-bit items of data, each item having the value 0 or 1
- Two advantages to bit-oriented view:
  - Memory can be used most efficiently for storing an array of Boolean or binary data items in which each item can take on only the values 1 (true) and 0 (false)
  - To manipulate the bits of a data item
    - If floating-point operations are implemented in software, we need to be able to shift significant bits in some operations
    - To convert from IRA to packed decimal, we need to extract the rightmost 4 bits of each byte

x86 Data Types

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value contained in a byte, word, or doubleword, using twos complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2^{23} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2^{23} - 1$ bytes.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types.

x86 Numeric Data Formats



Byte unsigned integer

Word unsigned integer

Doubleword unsigned integer

Quadword unsigned integer

Byte signed integer
(twos complement)

Word signed integer
(twos complement)

Doubleword signed integer
(twos complement)

Quadward usigned integer
(twos complement)t

Half precision
floating point

Single precision
floating point

Double precision
floating point

Double extended precision
floating point

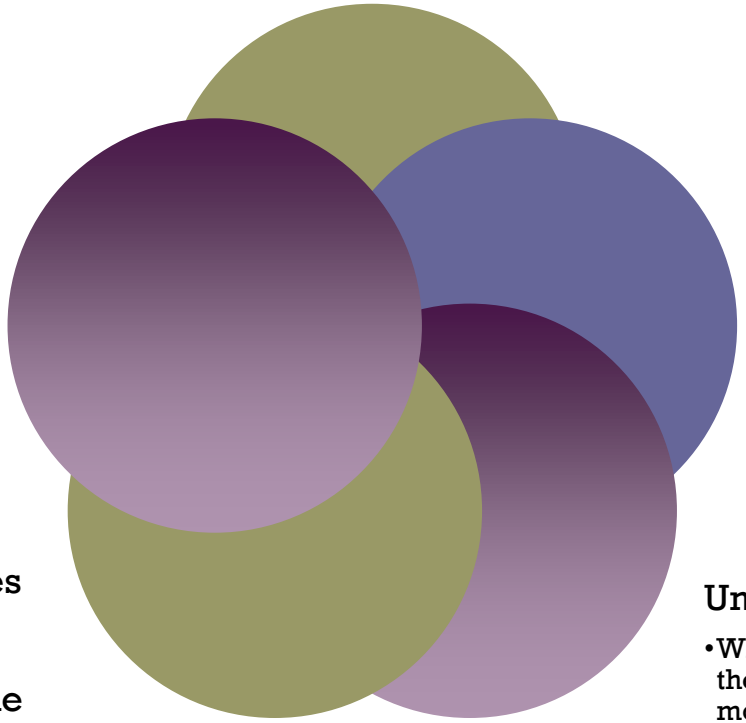
Single-Instruction-Multiple-Data (SIMD) Data Types

- Introduced to the x86 architecture as part of the extensions of the instruction set to optimize performance of multimedia applications
- These extensions include MMX (multimedia extensions) and SSE (streaming SIMD extensions)
- The basic concept is that multiple operands are packed into a single referenced memory item and that these multiple operands are operated on in parallel
- Data types:
  - Packed byte and packed byte integer
  - Packed word and packed word integer
  - Packed doubleword and packed doubleword integer
  - Packed quadword and packed quadword integer
  - Packed single-precision floating-point and packed double-precision floating-point

ARM Data Types

ARM processors support data types of:

- 8 (byte)
- 16 (halfword)
- 32 (word) bits in length



All three data types can also be used for twos complement signed integers

Alignment checking

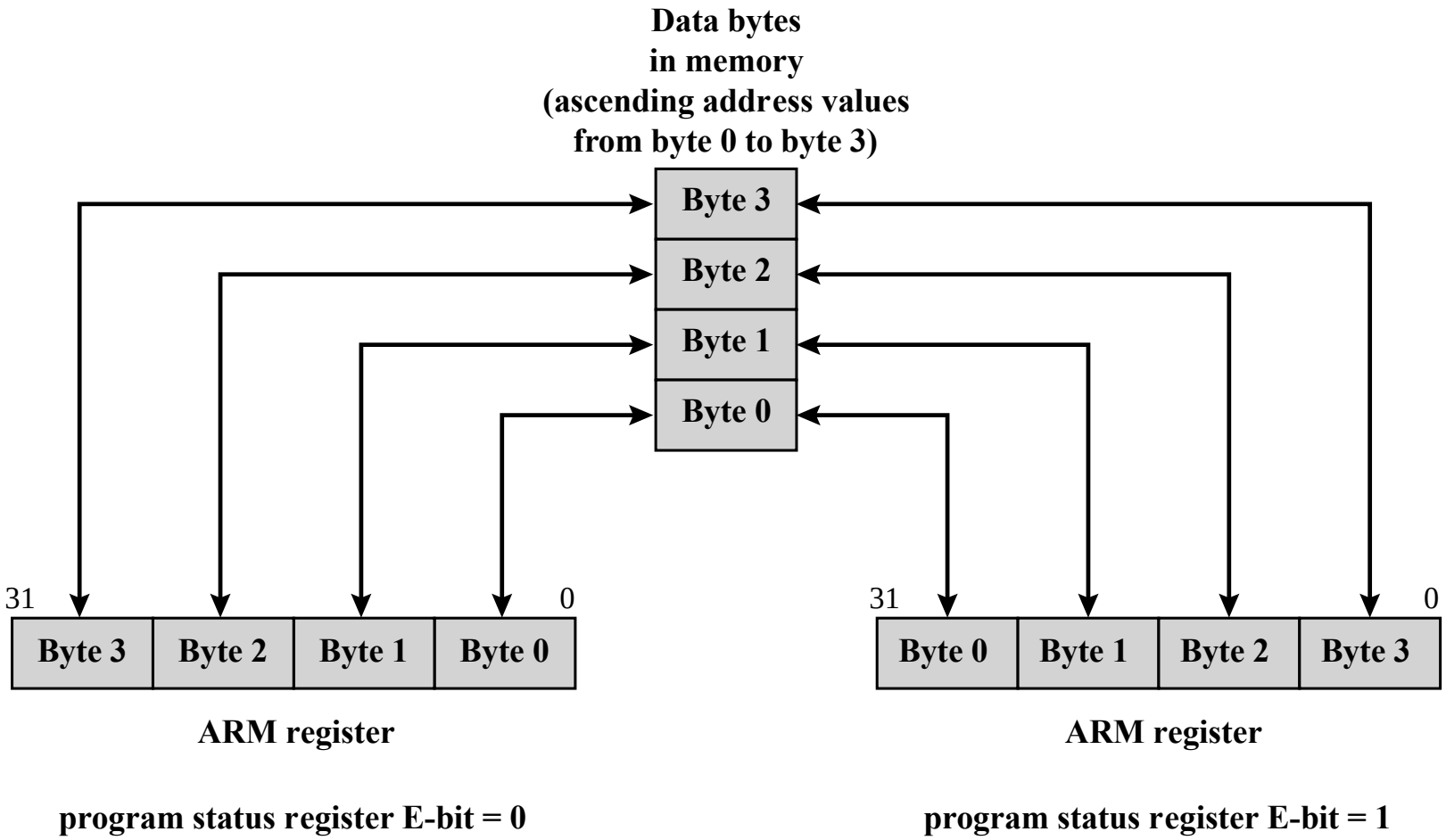
- When the appropriate control bit is set, a data abort signal indicates an alignment fault for attempting unaligned access

For all three data types an unsigned interpretation is supported in which the value represents an unsigned, nonnegative integer

Unaligned access

- When this option is enabled, the processor uses one or more memory accesses to generate the required transfer of adjacent bytes transparently to the programmer

ARM Endian Support—Word Load/Store with E-Bit



Common x86 Instruction Set Operations (1 of 3)

(a) Data Transfer

Operation Name	Description
MOV Dest, Source	Move data between registers or between register and memory or immediate to register.
XCHG Op1, Op2	Swap contents between two registers or register and memory.
PUSH Source	Decrements stack pointer (ESP register), then copies the source operand to the top of stack.
POP Dest	Copies top of stack to destination and increments ESP.

(b) Arithmetic

Operation Name	Description
ADD Dest, Source	Adds the destination and the source operand and stores the result in the destination. Destination can be register or memory. Source can be register, memory, or immediate.
SUB Dest, Source	Subtracts the source from the destination and stores the result in the destination.
MUL Op	Unsigned integer multiplication of the operand by the AL, AX, or EAX register and stores in the register. Opcode indicates size of register.
IMUL Op	Signed integer multiplication.
DIV Op	Divides unsigned the value in the AX, DX:AX, EDX:EAX, or RDX:RAX registers (dividend) by the source operand (divisor) and stores the result in the AX (AH:AL), DX:AX, EDX:EAX, or RDX:RAX registers.
IDIV Op	Signed integer division.
INC Op	Adds 1 to the destination operand, while preserving the state of the CF flag.
DEC Op	Subtracts 1 from the destination operand, while preserving the state of the CF flag.
NEG Op	Replaces the value of operand with (0 – operand), using twos complement representation.
CMP Op1, Op2	Compares the two operands by subtracting the second operand from the first operand and sets the status flags in the EFLAGS register according to the results.

Common x86 Instruction Set Operations (2 of 3)

(c) Shift and Rotate

Operation Name	Description
SAL Op, Quantity	Shifts the source operand left by from 1 to 31 bit positions. Empty bit positions are cleared. The CF flag is loaded with the last bit shifted out of the operand.
SAR Op, Quantity	Shifts the source operand right by from 1 to 31 bit positions. Empty bit positions are cleared if the operand is positive and set if the operand is negative. The CF flag is loaded with the last bit shifted out of the operand.
SHR Op, Quantity	Shifts the source operand right by from 1 to 31 bit positions. Empty bit positions are cleared and the CF flag is loaded with the last bit shifted out of the operand.
ROL Op, Quantity	Rotate bits to the left, with wraparound. The CF flag is loaded with the last bit shifted out of the operand.
ROR Op, Quantity	Rotate bits to the right, with wraparound. The CF flag is loaded with the last bit shifted out of the operand.
RCL Op, Quantity	Rotate bits to the left, including the CF flag, with wraparound. This instruction treats the CF flag as a one-bit extension on the upper end of the operand.
RCR Op, Quantity	Rotate bits to the right, including the CF flag, with wraparound. This instruction treats the CF flag as a one-bit extension on the lower end of the operand.

(d) Logical

Operation Name	Description
NOT Op	Inverts each bit of the operand.
AND Dest, Source	Performs a bitwise AND operation on the destination and source operands and stores the result in the destination operand.
OR Dest, Source	Performs a bitwise OR operation on the destination and source operands and stores the result in the destination operand.
XOR Dest, Source	Performs a bitwise XOR operation on the destination and source operands and stores the result in the destination operand.
TEST Op1, Op2	Performs a bitwise AND operation on the two operands and sets the S, Z, and P status flags. The operands are unchanged.

Common x86 Instruction Set Operations (3 of 3)

(e) Transfer of Control

Operation Name	Description
CALL proc	Saves procedure linking information on the stack and branches to the called procedure specified using the operand. The operand specifies the address of the first instruction in the called procedure.
RET	Transfers program control to a return address located on the top of the stack. The return is made to the instruction that follows the CALL instruction.
JMP Dest	Transfers program control to a different point in the instruction stream without recording return information. The operand specifies the address of the instruction being jumped to.
Jcc Dest	Checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand. See Tables 13.8 and 13.9.
NOP	This instruction performs no operation. It is a one-byte or multi-byte NOP that takes up space in the instruction stream but does not impact machine context, except for the EIP register.
HLT	Stops instruction execution and places the processor in a HALT state. An enabled interrupt, a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal will resume execution.
WAIT	Causes the processor to repeatedly check for and handle pending, unmasked, floating-point exceptions before proceeding.
INT Nr	Interrupts current program, runs specified interrupt program

(f) Input/Output

Operation Name	Description
IN Dest, Source	Copies the data from the I/O port specified by the source operand to the destination operand, which is a register location.
INS Dest, Source	Copies the data from the I/O port specified by the source operand to the destination operand, which is a memory location.
OUT Dest, Source	Copies the byte, word, or doubleword value from the source register to the I/O port specified by the destination operand.
XOR Dest, Source	Copies byte, word, or doubleword from the source operand to the I/O port specified with the destination operand. The source operand is a memory location.

Processor Actions for Various Types of Operations

Data transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

Data Transfer

Most fundamental type of machine instruction

Must specify:

- Location of the source and destination operands
- The length of data to be transferred must be indicated
- The mode of addressing for each operand must be specified

Examples of IBM EAS/390 Data Transfer Operations

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (short)	32	Transfer from floating-point register to floating-point register
LE	Load (short)	32	Transfer from memory to floating-point register
LDR	Load (long)	64	Transfer from floating-point register to floating-point register
LD	Load (long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (short)	32	Transfer from floating-point register to memory
STD	Store (long)	64	Transfer from floating-point register to memory

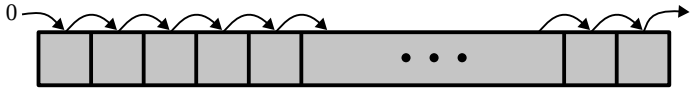
Arithmetic

- Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide
- These are provided for signed integer (fixed-point) numbers
- Often they are also provided for floating-point and packed decimal numbers
- Other possible operations include a variety of single-operand instructions:
  - Absolute
    - Take the absolute value of the operand
  - Negate
    - Negate the operand
  - Increment
    - Add 1 to the operand
  - Decrement
    - Subtract 1 from the operand

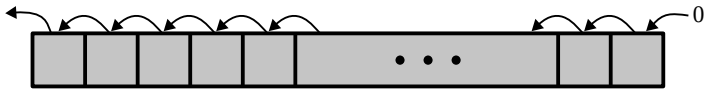
Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

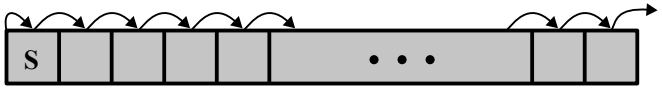
Shift and Rotate Operations



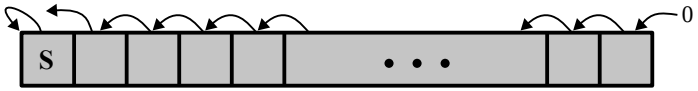
(a) Logical right shift



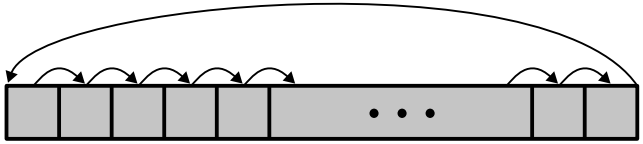
(b) Logical left shift



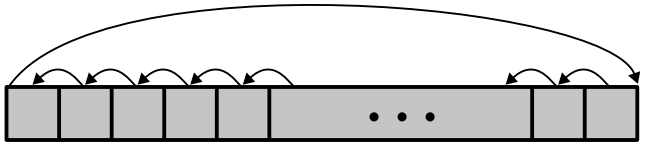
(c) Arithmetic right shift



(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

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

Conversion

Instructions that change the format or operate on the format of data

An example is converting from decimal to binary

An example of a more complex editing instruction is the EAS/390 Translate (TR) instruction

Input/Output

- Variety of approaches taken:
  - Isolated programmed I/O
  - Memory-mapped programmed I/O
  - DMA
  - Use of an I/O processor
- Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words

System Control

Instructions that can be executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory

Typically these instructions are reserved for the use of the operating system

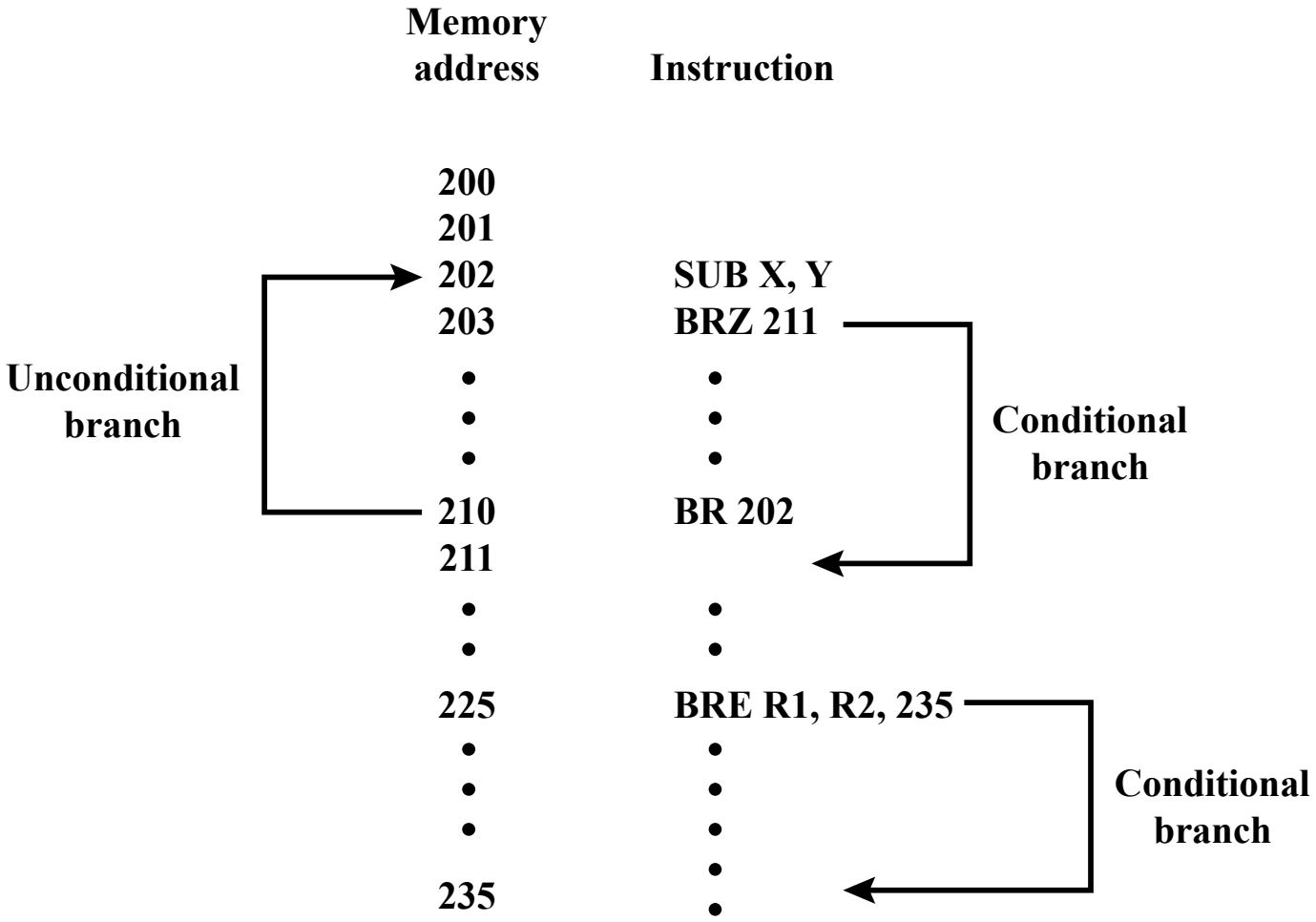
Examples of system control operations:

- A system control instruction may read or alter a control register
- An instruction to read or modify a storage protection key
- Access to process control blocks in a multiprogramming system

Transfer of Control

- Reasons why transfer-of-control operations are required:
  - It is essential to be able to execute each instruction more than once
  - Virtually all programs involve some decision making
  - It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time
- Most common transfer-of-control operations found in instruction sets:
  - Branch
  - Skip
  - Procedure call

Branch Instructions



BRE R1, R2, X : Branch to X if contents of R1 = contents of R2.
BRZ X : Branch to location X if result is zero.

Skip Instructions

Includes an implied address

Typically implies that one instruction be skipped, thus the implied address equals the address of the next instruction plus one instruction length

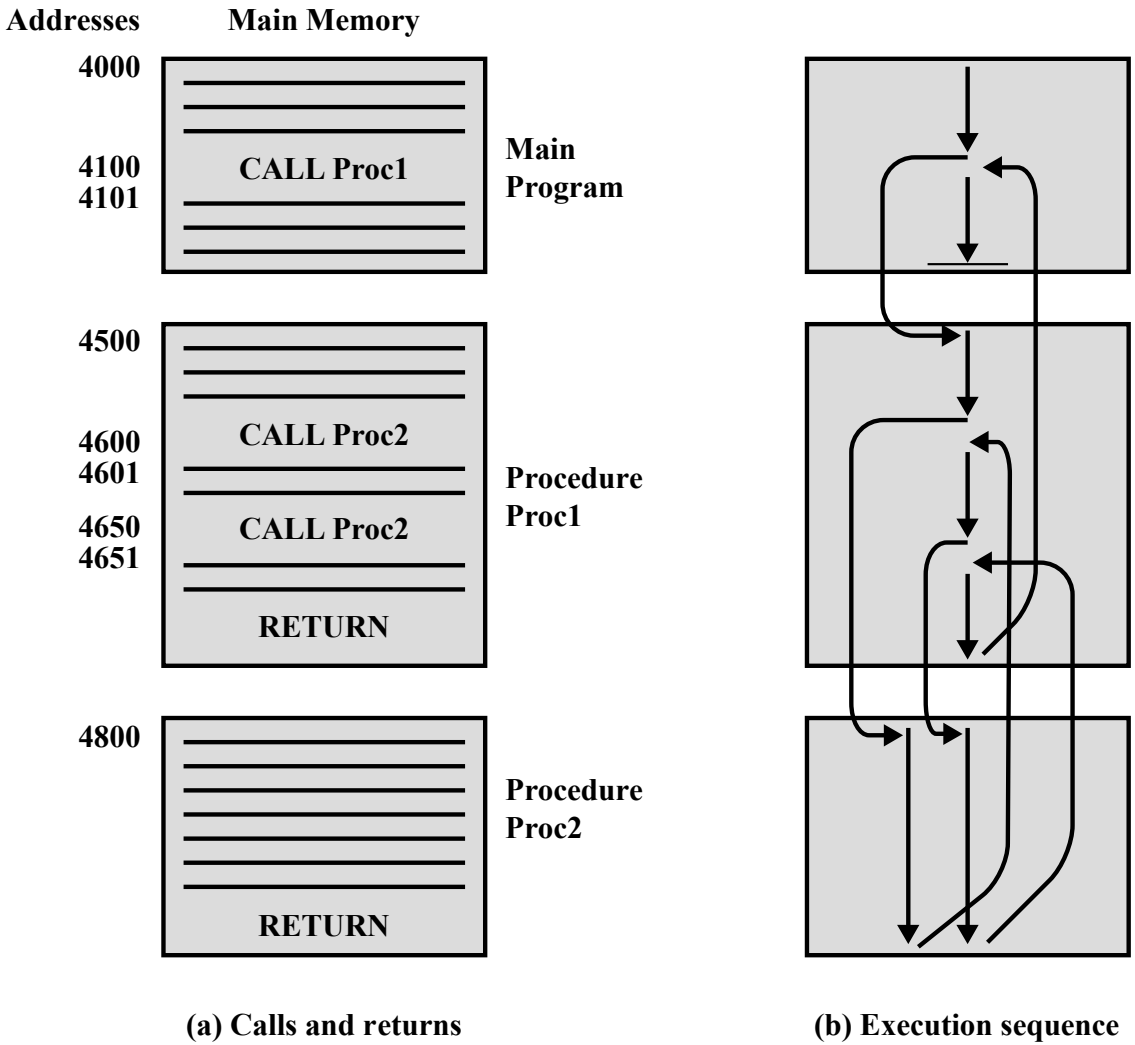
Because the skip instruction does not require a destination address field it is free to do other things

Example is the increment-and-skip-if-zero (ISZ) instruction

Procedure Call Instructions

- Self-contained computer program that is incorporated into a larger program
  - At any point in the program the procedure may be invoked, or *called*
  - Processor is instructed to go and execute the entire procedure and then return to the point from which the call took place
- Two principal reasons for use of procedures:
  - Economy
    - A procedure allows the same piece of code to be used many times
  - Modularity
- Involves two basic instructions:
  - A call instruction that branches from the present location to the procedure
  - Return instruction that returns from the procedure to the place from which it was called

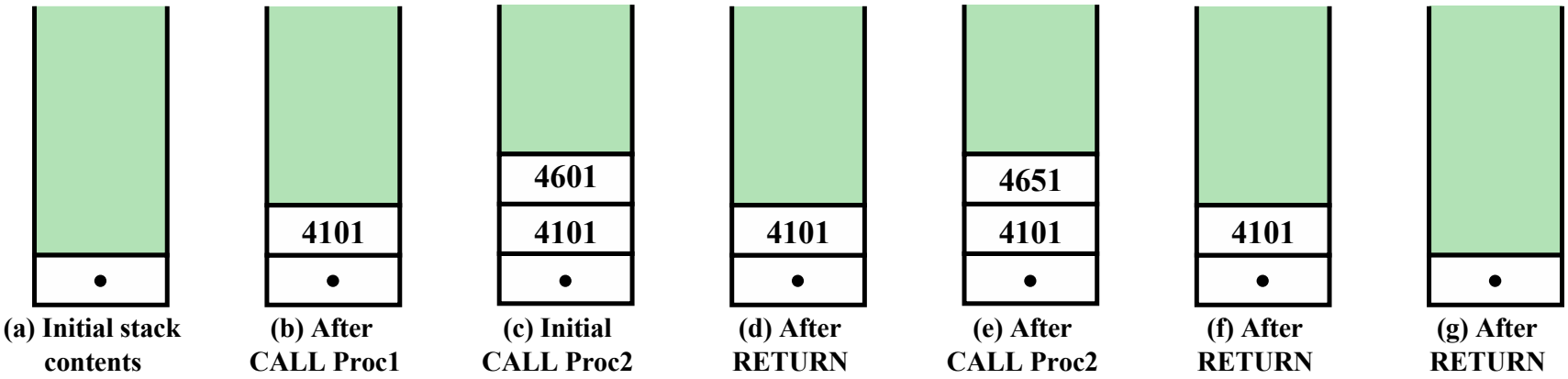
Nested Procedures



(a) Calls and returns

(b) Execution sequence

Use of Stack to Implement Nested Subroutines of Nested Procedures



x86 Operation Types

- The x86 provides a complex array of operation types including a number of specialized instructions
- The intent was to provide tools for the compiler writer to produce optimized machine language translation of high-level language programs
- Provides four instructions to support procedure call/return:
  - CALL
  - ENTER
  - LEAVE
  - RETURN
- When a new procedure is called the following must be performed upon entry to the new procedure:
  - Push the return point on the stack
  - Push the current frame pointer on the stack
  - Copy the stack pointer as the new value of the frame pointer
  - Adjust the stack pointer to allocate a frame

x86 Status Flags

Status Bit	Name	Description
C	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
P	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
A	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
Z	Zero	Indicates that the result of an arithmetic or logic operation is 0.
S	Sign	Indicates the sign of the result of an arithmetic or logic operation.
O	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

x86 Condition Codes for Conditional Jump and SETcc Instructions

Symbol	Condition Tested	Comment
A, NBE	C = 0 AND Z = 0	Above; Not below or equal (greater than, unsigned)
AE, NB, NC	C = 0	Above or equal; Not below (greater than or equal, unsigned); Not carry
B, NAE, C	C = 1	Below; Not above or equal (less than, unsigned); Carry set
BE, NA	C = 1 OR Z = 1	Below or equal; Not above (less than or equal, unsigned)
E, Z	Z = 1	Equal; Zero (signed or unsigned)
G, NLE	[(S = 1 AND O = 1) OR (S = 0 AND O = 0)]AND[Z = 0]	Greater than; Not less than or equal (signed)
GE, NL	(S = 1 AND O = 1) OR (S = 0 AND O = 0)	Greater than or equal; Not less than (signed)
L, NGE	(S = 1 AND O = 0) OR (S = 0 AND O = 0)	Less than; Not greater than or equal (signed)
LE, NG	(S = 1 AND O = 0) OR (S = 0 AND O = 1) OR (Z = 1)	Less than or equal; Not greater than (signed)
NE, NZ	Z = 0	Not equal; Not zero (signed or unsigned)
NO	O = 0	No overflow
NS	S = 0	Not sign (not negative)
NP, PO	P = 0	Not parity; Parity odd
O	O = 1	Overflow
P	P = 1	Parity; Parity even
S	S = 1	Sign (negative)

x86 Single-Instruction, Multiple-Data (SIMD) Instructions

- 1996 Intel introduced MMX technology into its Pentium product line
  - MMX is a set of highly optimized instructions for multimedia tasks
- Video and audio data are typically composed of large arrays of small data types
- Three new data types are defined in MMX
  - Packed byte
  - Packed word
  - Packed doubleword
- Each data type is 64 bits in length and consists of multiple smaller data fields, each of which holds a fixed-point integer

MMX Instruction Set

Category	Instruction	Description
Arithmetic	PADD [B, W, D]	Parallel add of packed eight bytes, four 16-bit words, or two 32-bit doublewords, with wraparound.
	PADDS [B, W]	Add with saturation.
	PADDUS [B, W]	Add unsigned with saturation.
	PSUB [B, W, D]	Subtract with wraparound.
	PSUBS [B, W]	Subtract with saturation.
	PSUBUS [B, W]	Subtract unsigned with saturation.
	PMULHW	Parallel multiply of four signed 16-bit words, with high-order 16 bits of 32-bit result chosen.
	PMULLW	Parallel multiply of four signed 16-bit words, with low-order 16 bits of 32-bit result chosen.
	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.
Comparison	PCMPEQ [B, W, D]	Parallel compare for equality; result is mask of 1s if true or 0s if false.
	PCMPGT [B, W, D]	Parallel compare for greater than; result is mask of 1s if true or 0s if false.
Conversion	PACKUSWB	Pack words into bytes with unsigned saturation.
	PACKSS [WB, DW]	Pack words into bytes, or doublewords into words, with signed saturation.
	PUNPCKH [BW, WD, DQ]	Parallel unpack (interleaved merge) high-order bytes, words, or doublewords from MMX register.
	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.
Logical	PAND	64-bit bitwise logical AND
	PNDN	64-bit bitwise logical AND NOT
	POR	64-bit bitwise logical OR
	PXOR	64-bit bitwise logical XOR
Shift	PSLL [W, D, Q]	Parallel logical left shift of packed words, doublewords, or quadword by amount specified in MMX register or immediate value.
	PSRL [W, D, Q]	Parallel logical right shift of packed words, doublewords, or quadword.
	PSRA [W, D]	Parallel arithmetic right shift of packed words, doublewords, or quadword.
Data transfer	MOV [D, Q]	Move doubleword or quadword to/from MMX register.
Statemgt	EMMS	Empty MMX state (empty FP registers tag bits).

Note: If an instruction supports multiple data types [byte (B), word (W), doubleword (D), quadword (Q)], the data types are indicated in brackets.

ARM Operation Types

- Load and store instructions
- Branch instructions
- Data-processing instructions
- Multiply instructions
- Parallel addition and subtraction instructions
- Extend instructions
- Status register access instructions

ARM Conditions for Conditional Instruction Execution

Code	Symbol	Condition Tested	Comment
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS/HS	C = 1	Carry set/unsigned higher or same
0011	CC/LO	C = 0	Carry clear/unsigned lower
00100	MI	N = 1	Minus/negative
00101	PL	N = 0	Plus/positive or zero
00110	VS	V = 1	Overflow
00111	VC	V = 0	No overflow
1000	HI	C = 1 AND Z = 0	Unsigned higher
1001	LS	C = 0 OR Z = 1	Unsigned lower or same
1010	GE	N = V [(N = 1 AND V = 1) OR (N = 0 AND V = 0)]	Signed greater than or equal
1011	LT	N ≠ V [(N = 1 AND V = 0) OR (N = 0 AND V = 1)]	Signed less than
1100	GT	(Z = 0) AND (N = V)	Signed greater than
1101	LE	(Z = 1) OR (N ≠ V)	Signed less than or equal
1110	AL	–	Always (unconditional)
1111	–	–	This instruction can only be executed unconditionally