

SEN361 Computer Organization

Prof. Dr. Hasan Hüseyin BALIK
(6th Week)



Outline

3. The Central Processing Unit

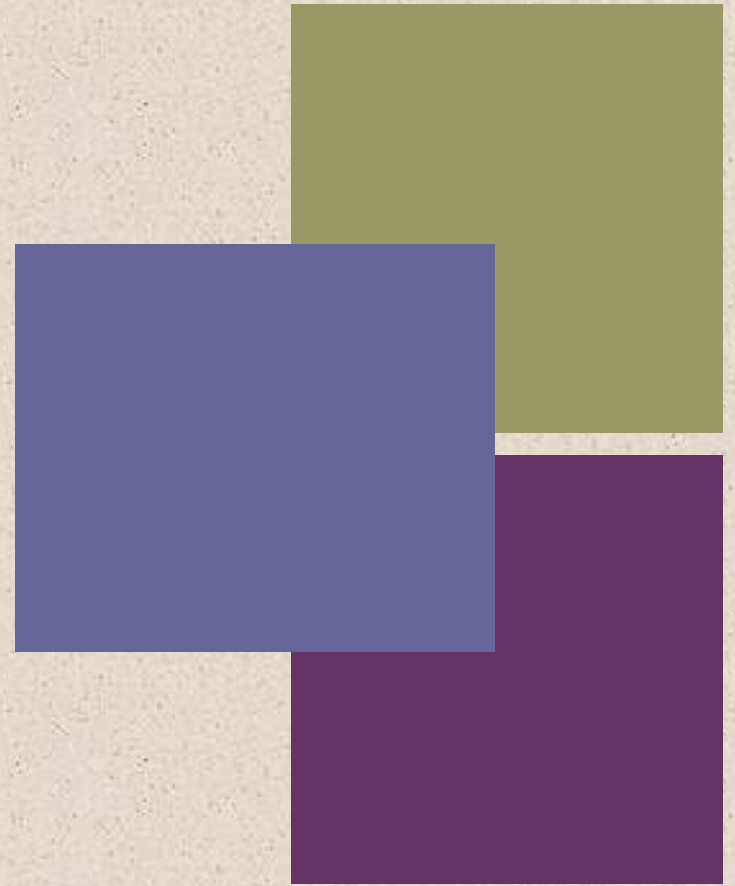
3.1 Instruction Sets: Characteristics and Functions

3.2 Instruction Sets: Addressing Modes and Formats

3.3 Processor Structure and Function

3.4 Reduced Instruction Set Computers

3.5 Instruction-Level Parallelism and Superscalar Processors



+

3.1 Instruction Sets: Characteristics and Functions



3.1 Outline

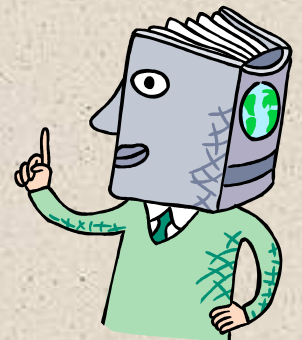
- Machine Instruction Characteristics
- Types of Operands
- Intel x86 and Arm Data Types
- Types of Operations
- Intel x86 and ARM Operation Types



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



Elements of a Machine Instruction

Operation code (opcode)

- Specifies the operation to be performed. The operation is specified by a binary code, known as the operation code, or *opcode*

Source operand reference

- The operation may involve one or more source operands, that is, 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 after the execution of this instruction is complete

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

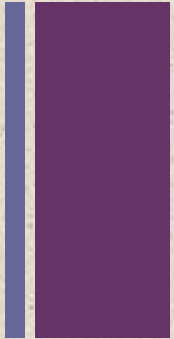
- A processor contains one or more registers that may be referenced by machine instructions.
- If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

4) Immediate

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



Instruction Representation



- 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

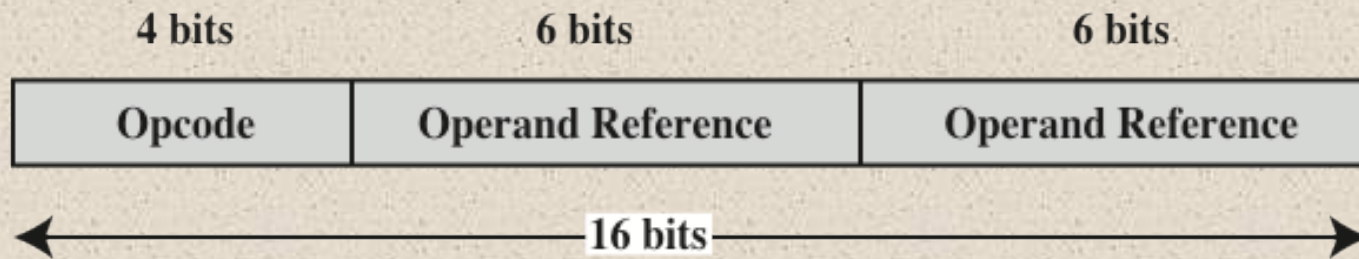


Figure 12.2 A Simple Instruction Format

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

Data
processing

Data
storage

Control

Data
movement

- 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

+ Number of Addresses

Instruction	Comment
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction	Comment
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

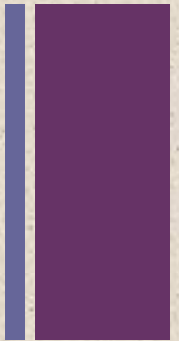
Instruction	Comment
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

(c) One-address instructions

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



Utilization of Instruction Addresses (Nonbranching Instructions)



Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	$A \leftarrow B \text{ OP } C$
2	OP A, B	$A \leftarrow A \text{ OP } B$
1	OP A	$AC \leftarrow AC \text{ OP } A$
0	OP	$T \leftarrow (T - 1) \text{ OP } T$

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

- How many and which operations to provide and how complex operations should be

Data types

- The various types of data upon which operations are performed

Instruction format

- Instruction length in bits, number of addresses, size of various fields, etc.

Registers

- Number of processor registers that can be referenced by instructions and their use

Addressing

- The mode or modes by which the address of an operand is specified

Types of Operands

Addresses

Numbers

Characters

Logical Data

+ 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 $2_{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2_{32} - 1$ bytes.
Floating point	See Figure 12.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types



x86 Numeric Data Formats

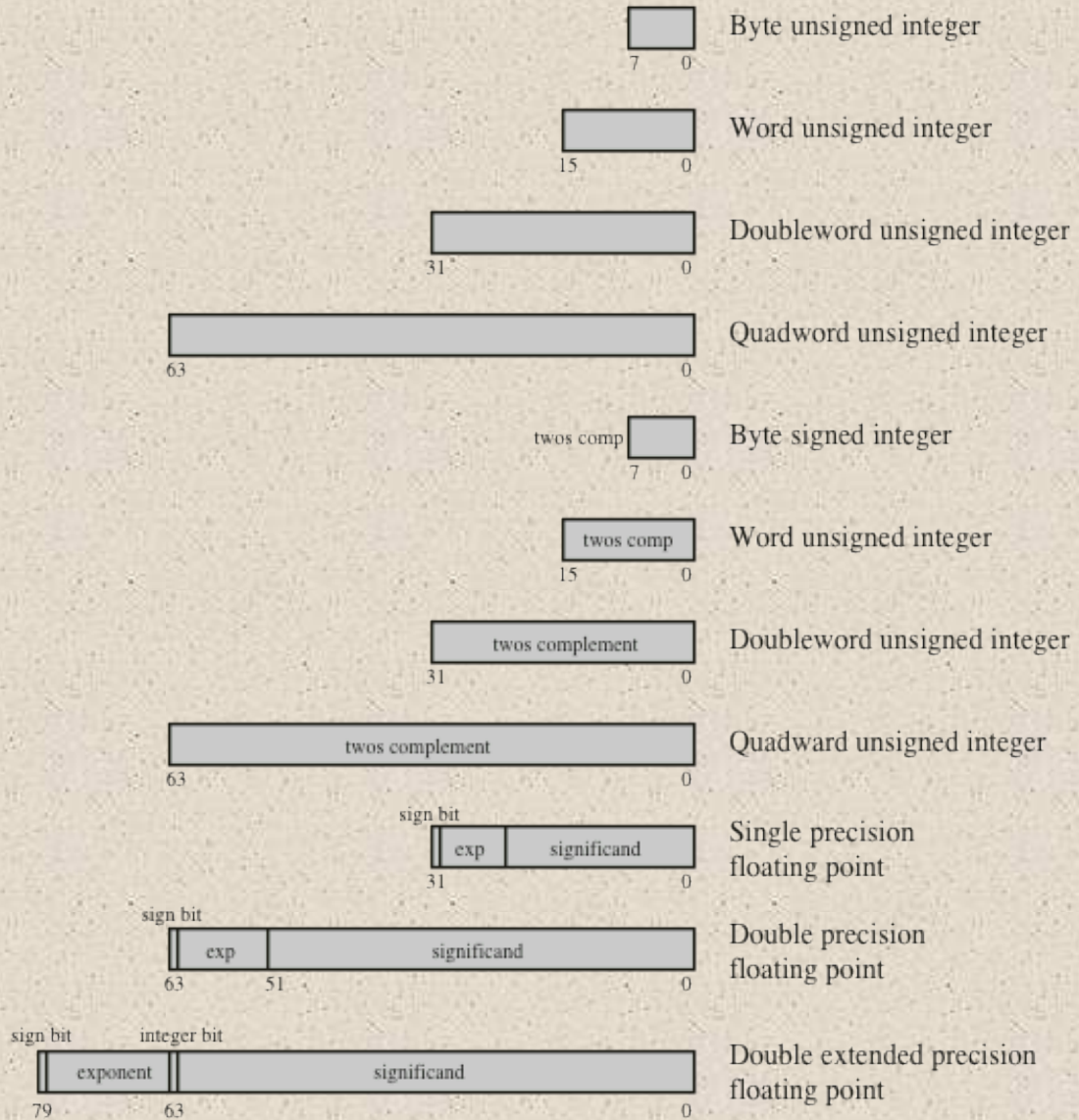


Figure 12.4 x86 Numeric Data Formats



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)
- 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

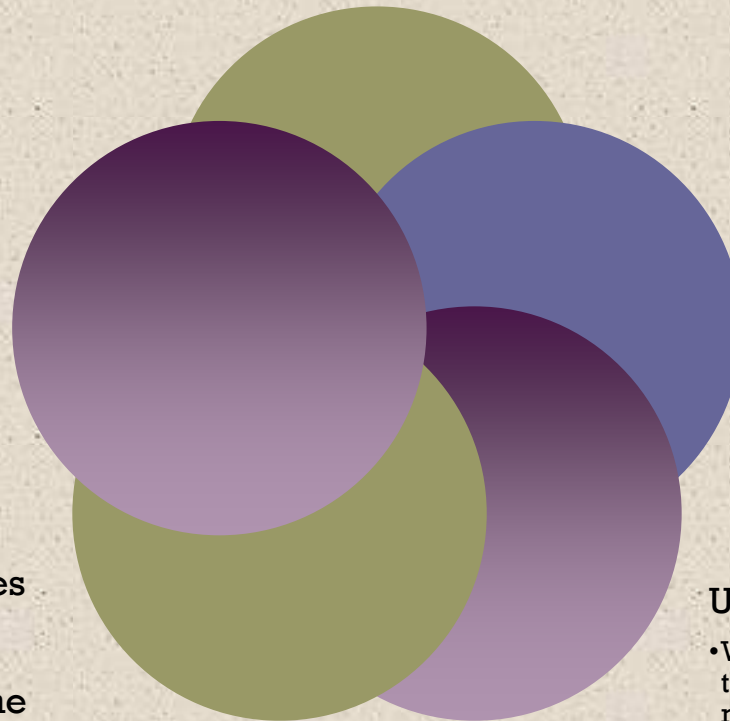


ARM processors support data types of:

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

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

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



Alignment checking

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

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

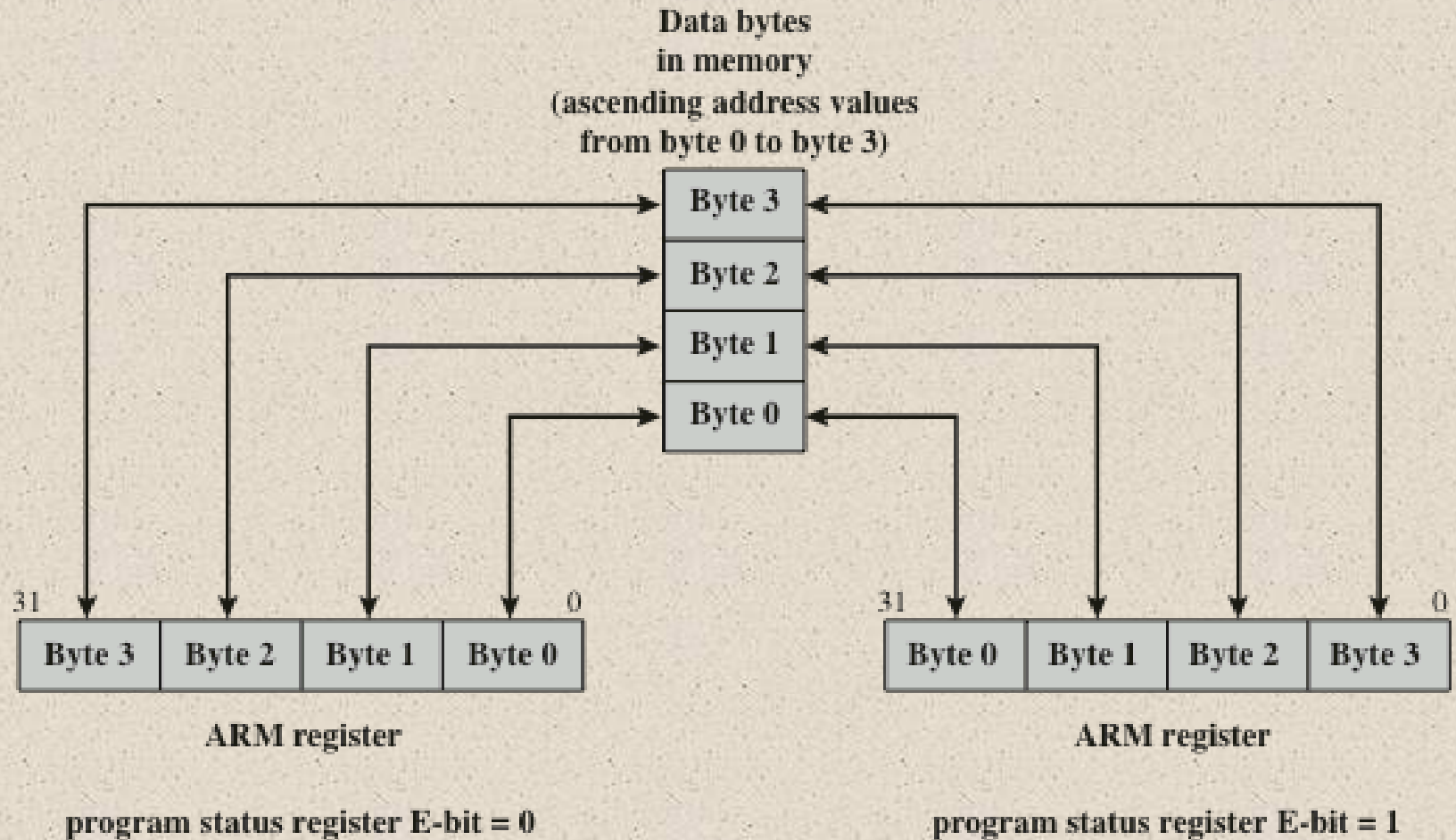


Figure 12.5 ARM Endian Support - Word Load/Store with E-bit



Common Instruction Set Operations

(page 1 of 2)

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
	Add	Compute sum of two operands
Arithmetic	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
OR		Perform logical OR
NOT (complement)		Perform logical NOT
Exclusive-OR		Perform logical XOR
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 of 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



Common Instruction Set Operations

(page 2 of 2)

Type	Operation Name	Description
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)

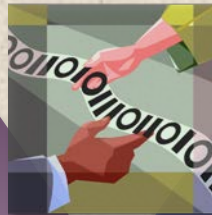
Processor Actions for Various Types of Operations

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:

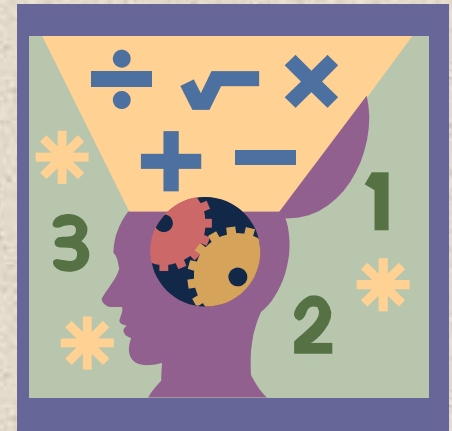
- Location of the source and destination operands
- The length of data to be transferred must be indicated
- The mode of addressing for each operand must be specified

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



- 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



Arithmetic



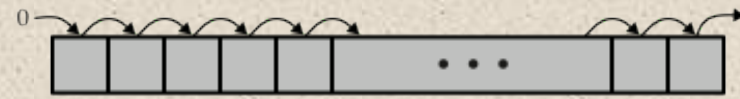
Logical

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

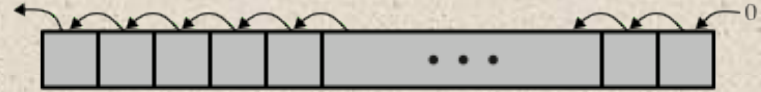
Table 12.6 Basic Logical Operations



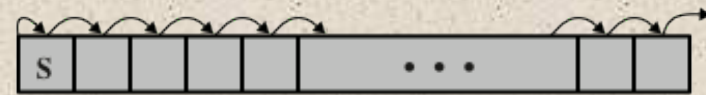
Shift and Rotate Operations



(a) Logical right shift



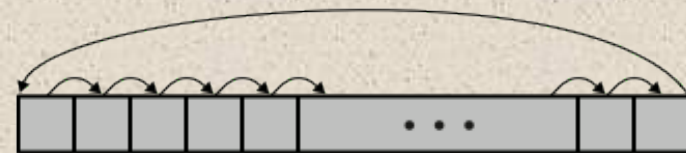
(b) Logical left shift



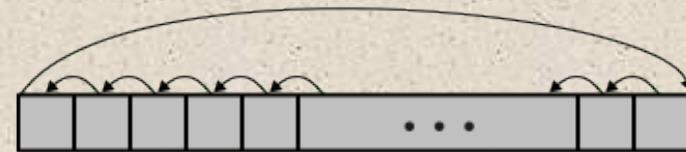
(c) Arithmetic right shift



(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

Figure 12.6 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



+

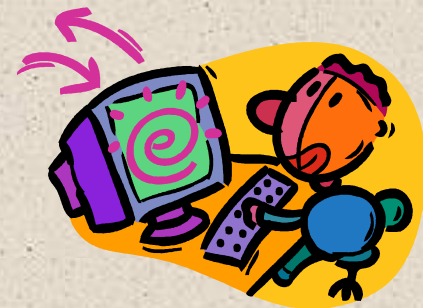
Examples of Shift and Rotate Operations



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 Instruction

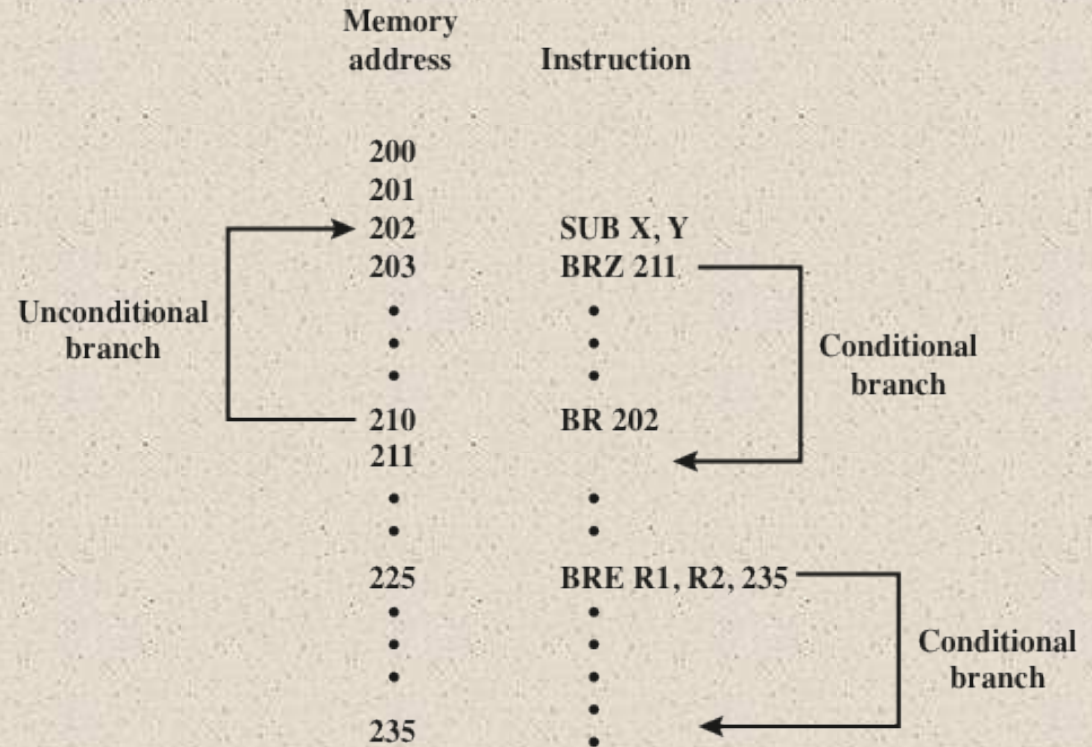


Figure 12.7 Branch Instructions

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



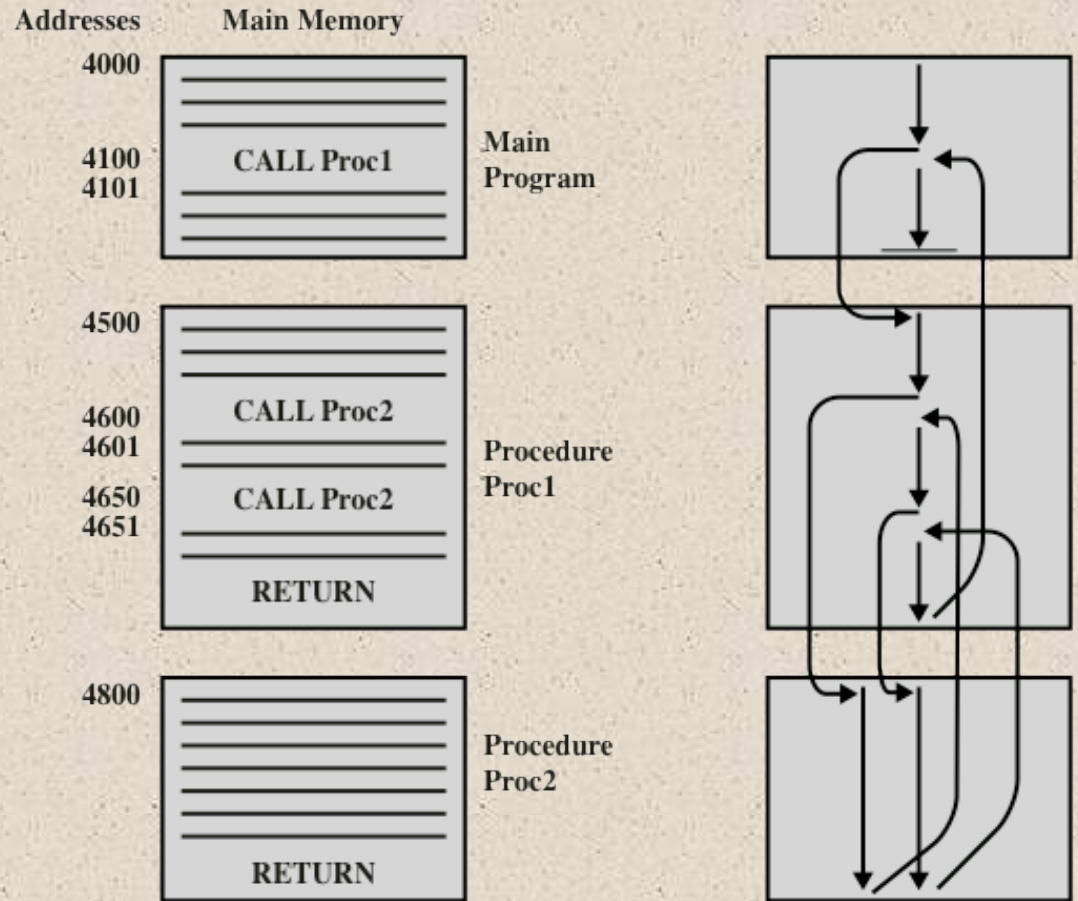
Procedure Call Instructions



- 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

Figure 12.8 Nested Procedures

Use of Stack to Implement Nested Procedures

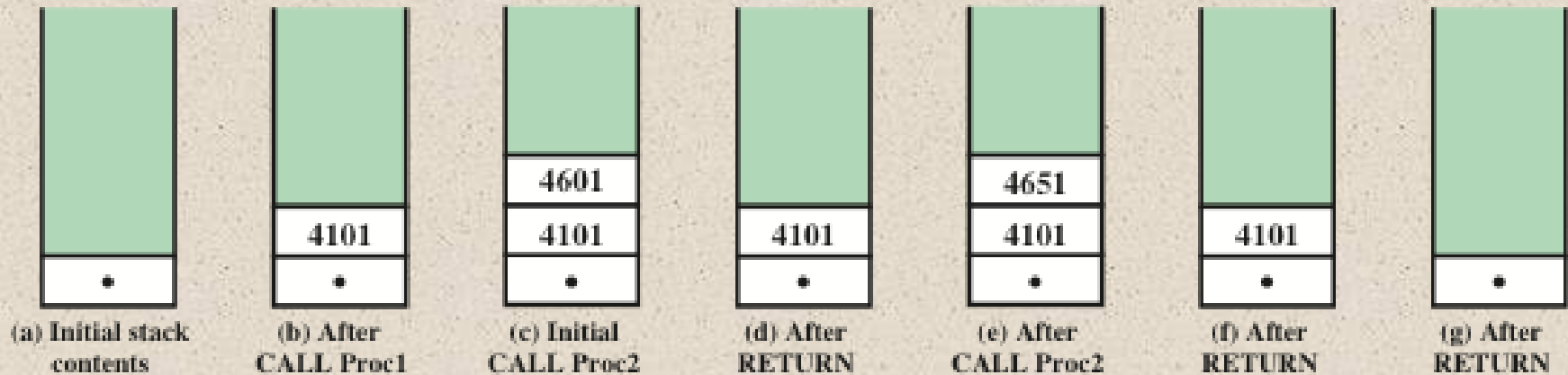



Figure 12.9 Use of Stack to Implement Nested Procedures of Figure 12.8

Instruction	Description
Data Movement	
MOV	Move operand, between registers or between register and memory.
PUSH	Push operand onto stack.
PUSHA	Push all registers on stack.
MOVSX	Move byte, word, dword, sign extended. Moves a byte to a word or a word to a doubleword with twos-complement sign extension.
LEA	Load effective address. Loads the offset of the source operand, rather than its value to the destination operand.
XLAT	Table lookup translation. Replaces a byte in AL with a byte from a user-coded translation table. When XLAT is executed, AL should have an unsigned index to the table. XLAT changes the contents of AL from the table index to the table entry.
IN, OUT	Input, output operand from I/O space.
Arithmetic	
ADD	Add operands.
SUB	Subtract operands.
MUL	Unsigned integer multiplication, with byte, word, or double word operands, and word, doubleword, or quadword result.
IDIV	Signed divide.
Logical	
AND	AND operands.
BTS	Bit test and set. Operates on a bit field operand. The instruction copies the current value of a bit to flag CF and sets the original bit to 1.
BSF	Bit scan forward. Scans a word or doubleword for a 1-bit and stores the number of the first 1-bit into a register.
SHL/SHR	Shift logical left or right.
SAL/SAR	Shift arithmetic left or right.
ROL/ROR	Rotate left or right.
SETcc	Sets a byte to zero or one depending on any of the 16 conditions defined by status flags.
Control Transfer	
JMP	Unconditional jump.
CALL	Transfer control to another location. Before transfer, the address of the instruction following the CALL is placed on the stack.
JE/JZ	Jump if equal/zero.
LOOPE/LOOPZ	Loops if equal/zero. This is a conditional jump using a value stored in register ECX. The instruction first decrements ECX before testing ECX for the branch condition.
INT/INTO	Interrupt/Interrupt if overflow. Transfer control to an interrupt service routine


x86
Operation
Types (With
Examples of
Typical
Operations)

(page 1 of 2)



String Operations	
MOVS	Move byte, word, dword string. The instruction operates on one element of a string, indexed by registers ESI and EDI. After each string operation, the registers are automatically incremented or decremented to point to the next element of the string.
LODS	Load byte, word, dword of string.
High-Level Language Support	
ENTER	Creates a stack frame that can be used to implement the rules of a block-structured high-level language.
LEAVE	Reverses the action of the previous ENTER.
BOUND	Check array bounds. Verifies that the value in operand 1 is within lower and upper limits. The limits are in two adjacent memory locations referenced by operand 2. An interrupt occurs if the value is out of bounds. This instruction is used to check an array index.
Flag Control	
STC	Set Carry flag.
LAHF	Load AH register from flags. Copies SF, ZF, AF, PF, and CF bits into A register.
Segment Register	
LDS	Load pointer into DS and another register.
System Control	
HLT	Halt.
LOCK	Asserts a hold on shared memory so that the Pentium has exclusive use of it during the instruction that immediately follows the LOCK.
ESC	Processor extension escape. An escape code that indicates the succeeding instructions are to be executed by a numeric coprocessor that supports high-precision integer and floating-point calculations.
WAIT	Wait until BUSY# negated. Suspends Pentium program execution until the processor detects that the BUSY pin is inactive, indicating that the numeric coprocessor has finished execution.
Protection	
SGDT	Store global descriptor table.
LSL	Load segment limit. Loads a user-specified register with a segment limit.
VERR/VERW	Verify segment for reading/writing.
Cache Management	
INVD	Flushes the internal cache memory.
WBINVD	Flushes the internal cache memory after writing dirty lines to memory.
INVLPG	Invalidates a translation lookaside buffer (TLB) entry.

**x86
Operation
Types (With
Examples of
Typical
Operations)
(page 2 of 2)**



Call/Return Instructions



- The x86 provides four instructions to support procedure call/return:
 - CALL
 - ENTER
 - LEAVE
 - RETURN
- Common means of implementing the procedure is via the use of stack frames
- The CALL instruction pushes the current instruction pointer value onto the stack and causes a jump to the entry point of the procedure by placing the address of the entry point in the instruction pointer
- The ENTER instruction was added to the instruction set to provide direct support for the compiler

x86 Status Flags

Status Bit	Name	Description
CF	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.
PF	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
AF	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
ZF	Zero	Indicates that the result of an arithmetic or logic operation is 0.
SF	Sign	Indicates the sign of the result of an arithmetic or logic operation.
OF	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.



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

x86 Condition Codes for Conditional Jump and SETcc Instructions



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.	
	PADDQ [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.	
Comparison	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.	
	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.	
Logical	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.	
	PAND	64-bit bitwise logical AND	
	PANDN	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.	
State Mgt	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
0100	MI	$N = 1$	Minus/negative
0101	PL	$N = 0$	Plus/positive or zero
0110	VS	$V = 1$	Overflow
0111	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	$N \neq V$ [($N = 1$ 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 $(N \neq V)$	Signed less than or equal
1110	AL	—	Always (unconditional)
1111	—	—	This instruction can only be executed unconditionally