Performance, RISC & CISC

Topics:

- Performance Measure
- Complex Instruction Set Computers (CISC)
- Reduced Instruction Set Computers (RISC)
- Characteristics Comparison
  - Size, length and complexity of Instructions
  - Number and usage of registers
  - Effective address
    - Memory access
    - ALU operands

Performance Measure

$$T = (N * S) / R$$

Why Processor Time and not Wall Clock Time?

Work Load of Processor? Operating System?

- **T** = processor time to execute a program

High-Level or Assembly?

- **N** = number of machine instructions executed
- **S** = average number of basic steps (e.g., one clock cycle) to execute one machine instruction
- **R** = clock rate in cycles per second

Processor Design

$$T = (N * S) / R$$

- Design objective?
  - To reduce T

1. Make N smaller?

2. Make S smaller?

3. Make R larger?

- How?
  - All of the above
  - But have to make tradeoffs among N, S, and R

Clock Rate R

1. Clock rate R:
- Higher clock rate
- More cycles per second
- **How?**

1. Increase **R** through fabrication technology/material
2. Reduce the amount of processing done in one step

$$T = (N * S) / R$$

R = clock rate in cycles per second

N = number of machine instructions executed

S = average number of basic steps (e.g., one clock cycle) to execute one machine instruction

Instruction Set

$$T = (N * S) / R$$

2. Two types of Instruction sets:
   - RISC: Reduced Instruction Set Computer
     - Small or Large N
     - Small or Large S
   - CISC: Complex Instruction Set Computer
     - Small or Large N
     - Small or Large S

R = clock rate in cycles per second

N = number of machine instructions executed

S = average number of basic steps (one clock cycle each) to execute one machine instruction

Note relationship between N and S

Compiler

3. Optimizing compiler:
   - Reduce (N * S)
     - Which is better: RISC or CISC?
     - Think instruction complexity, dependency and side-effects

T = (N * S) / R

R = clock rate in cycles per second

N = number of machine instructions executed

S = average number of basic steps (one clock cycle each) to execute one machine instruction

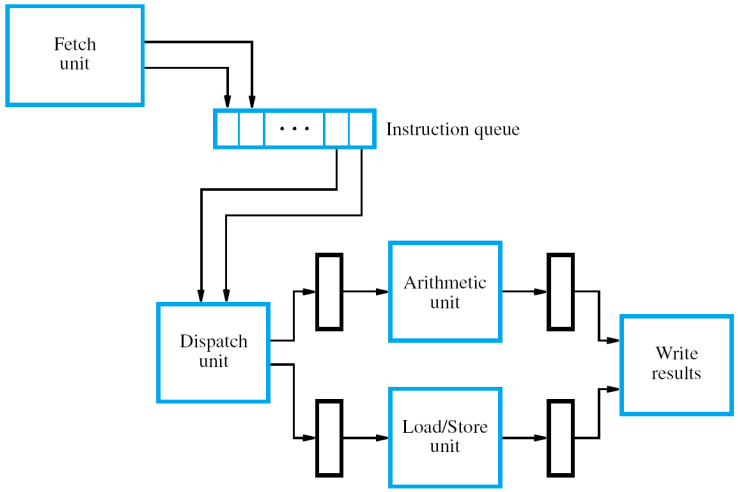
Co-design with hardware

- Compiler
- Operating System
- Firmware
  - Software embedded in hardware (ROM) that provides low-level control

Program with Assembly!

Parallelism

4. Two types of Parallelism:
   - Pipelining (assembly line)
     - It is Pseudo Parallelism
   - Resource replication
     - Multiple Instructions



$$T = (N * S) / R$$

R = clock rate in cycles per second

N = number of machine instructions executed

S = average number of basic steps (one clock cycle each) to execute one machine instruction

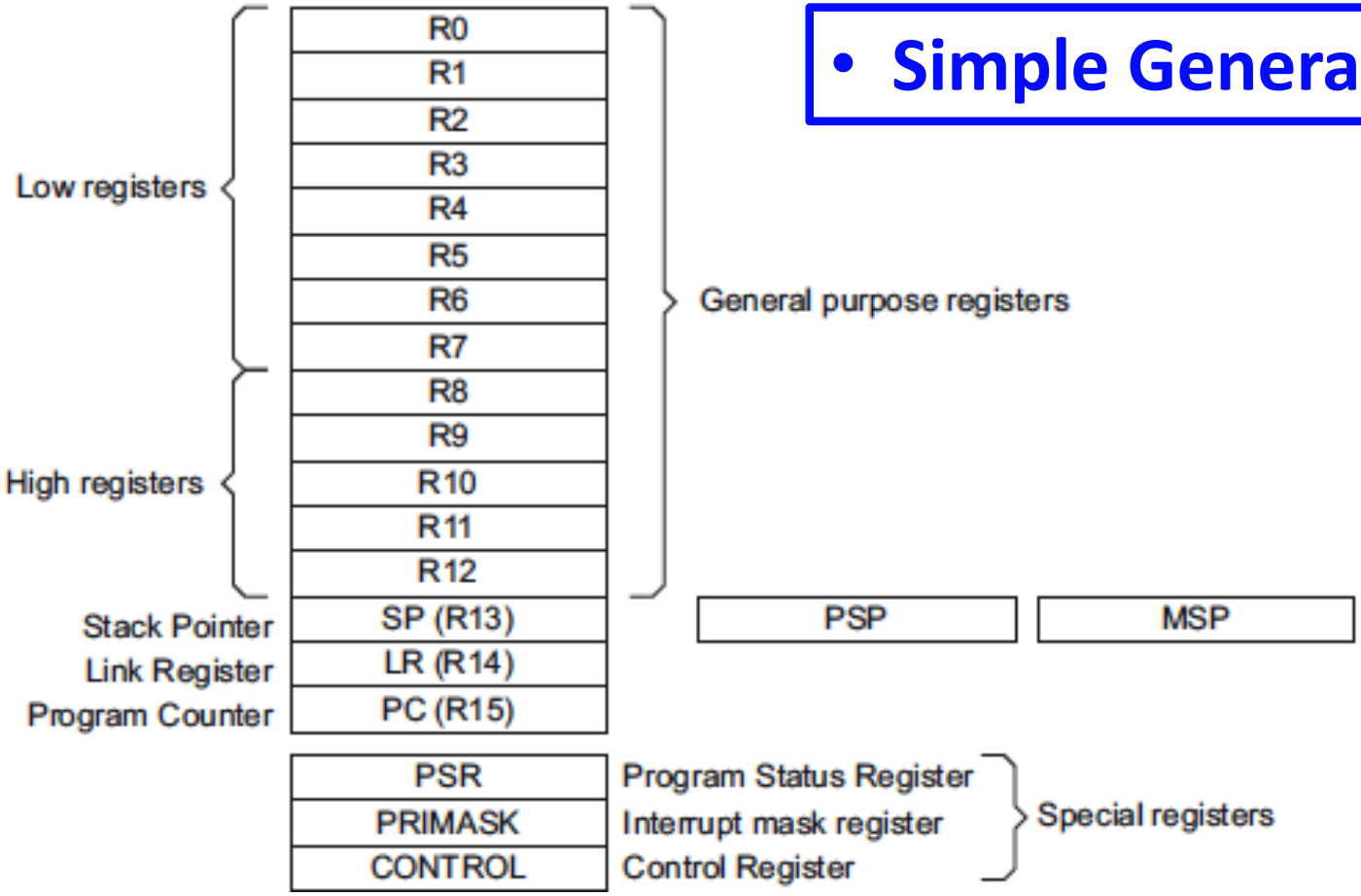
RISC & CISC

- Two fundamentally different ISA design approaches
  - **Complex Instruction Set Computers (CISC)**
    - Traditional approach since 1960s
    - Add-ons to previous generations
  - **Reduced Instruction Set Computers (RISC)**
    - 1980s research and development
    - then commercial products
  - Today's processors
    - Hybrid or RISC within a CISC

RISC vs CISC – Registers

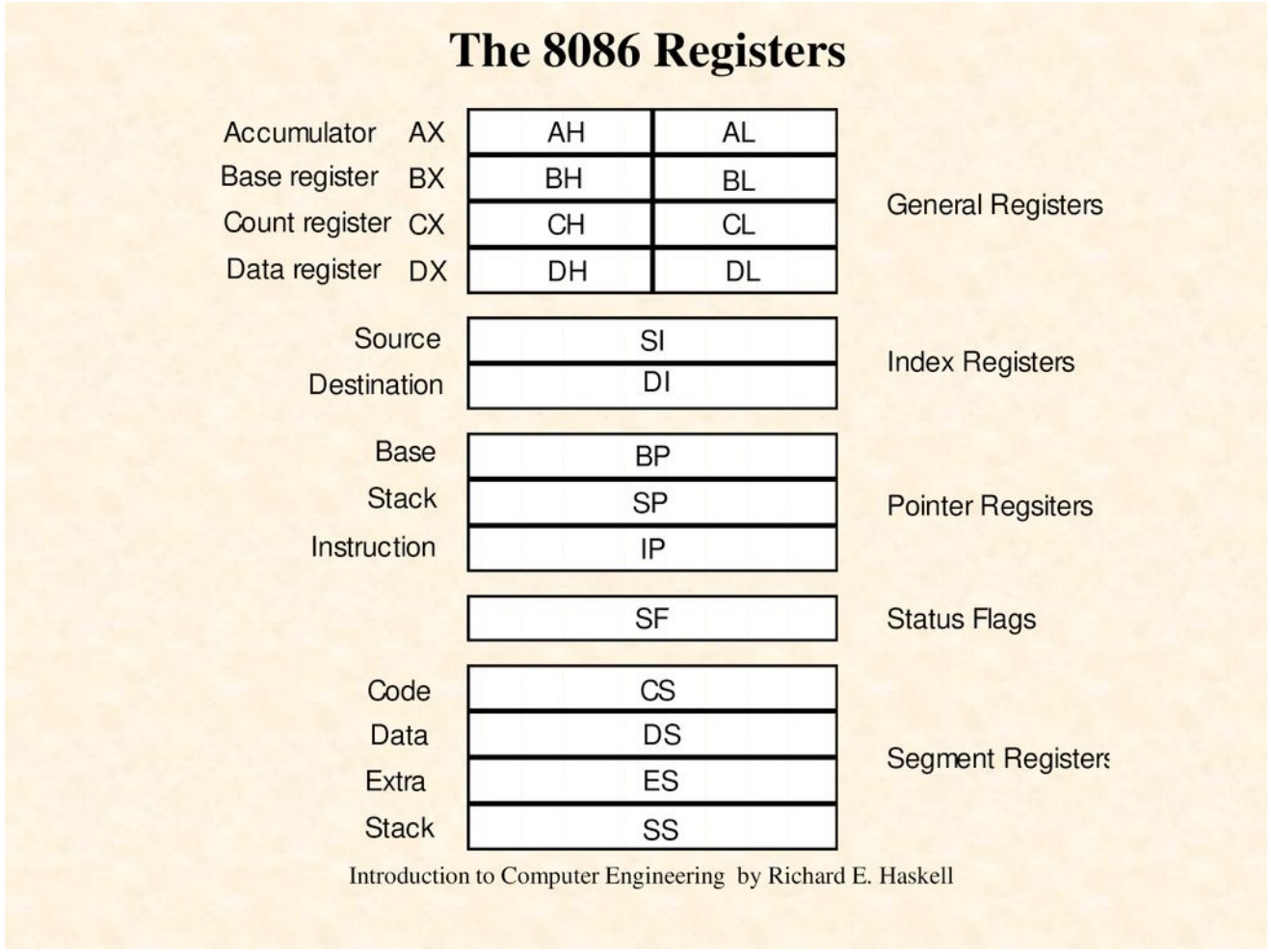
- Registers
  - RISC
    - Simple
    - General Purpose
  - CISC
    - More complex
    - Special Purpose

ARM Registers (RISC)



- **Simple General Purpose**

8086 Registers (CISC)

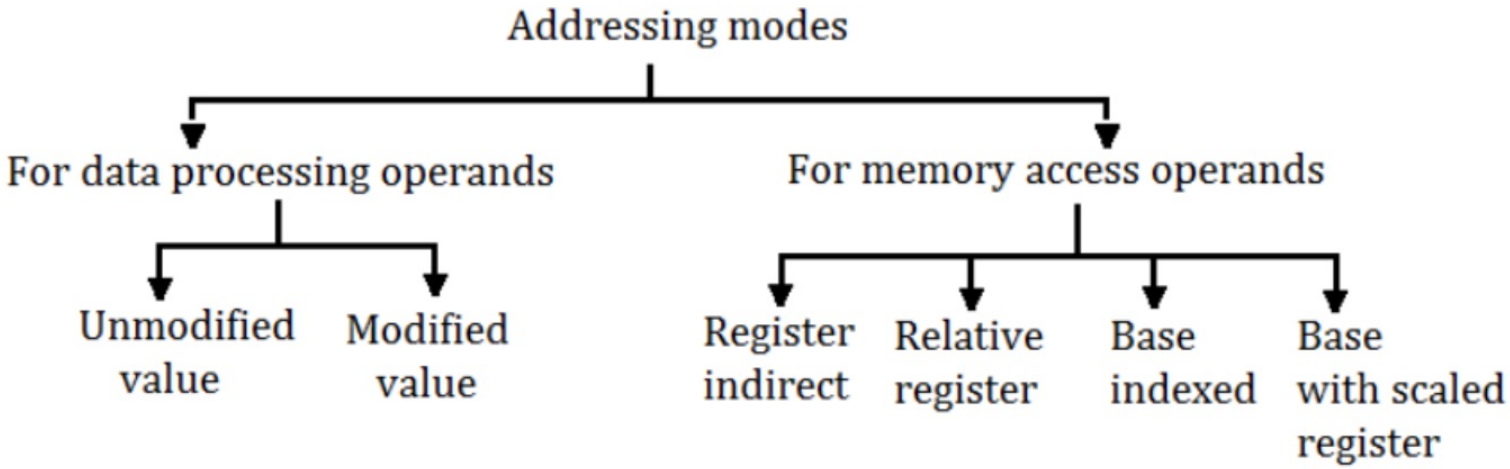


Special Purpose

RISC vs CISC – Effective Address

- Effective Address or Addressing Mode
  - **RISC**
    - Simple
    - Example: EA = X(Ri) where X is a constant
  - **CISC**
    - More complex
    - Example: EA = (X) where X is a memory location

ARM – Effective Address



8086 – Effective Address

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing

RISC vs CISC - ALU Operations

- Arithmetic and logic operations
  - **RISC**
    - Operands in registers or #imm
    - Example: Add R1, R2, #offset3
  - **CISC**
    - Operands in registers, memory, or #imm
    - Example: Add R1, (R3), #offset16

RISC vs CISC – Memory Access

- Memory Access
  - **RISC**
    - Load/Store Architecture
    - Example: Load R1,(R2); Store (R3),R1
  - **CISC**
    - Memory to Memory Transfer
    - Example: Move (R2),(R3)

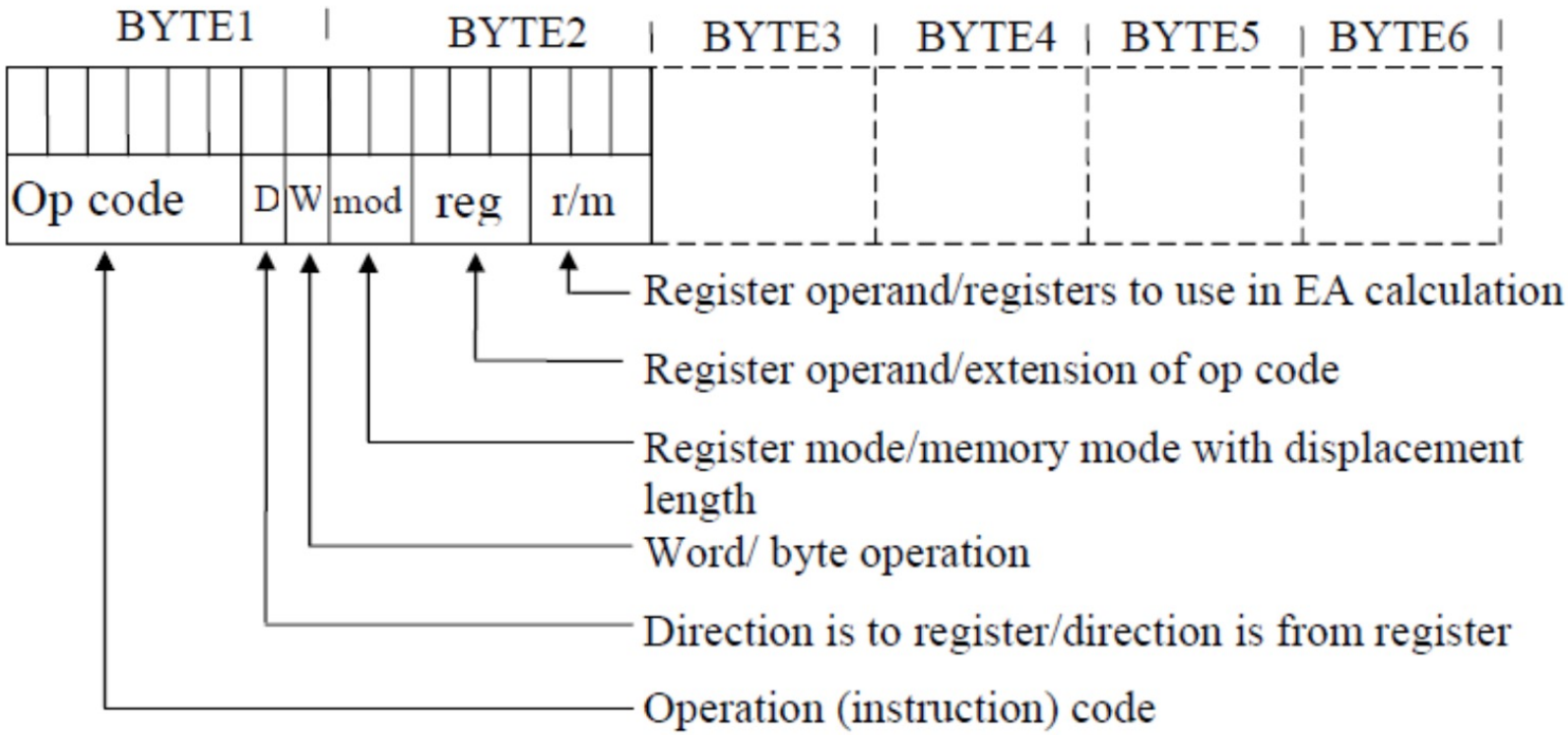
RISC vs CISC – Instruction Word Size

- Instruction Word
  - **RISC**
    - Instructions fit in a single word
    - Example: Range constraint in offset (3 or 8)
  - **CISC**
    - Instructions fit in one or more words
    - Example: Full address in second word

ARM – Instruction Word Size

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			*Move shifted register*
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			*Add/subtract*
3	0	0	1	Op		Rd		Offset8									*Move/compare/add /subtract immediate*
4	0	1	0	0	0	0	Op				Rs			Rd			*ALU operations*
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			*Hi register operations /branch exchange*
6	0	1	0	0	1	Rd			Word8								*PC-relative load*
7	0	1	0	1	L	B	0	Ro			Rb			Rd			*Load/store with register offset*
8	0	1	0	1	H	S	1	Ro			Rb			Rd			*Load/store sign-extended byte/halfword*
9	0	1	1	B	L	Offset5					Rb			Rd			*Load/store with immediate offset*
10	1	0	0	0	L	Offset5					Rb			Rd			*Load/store halfword*
11	1	0	0	1	L	Rd			Word8								*SP-relative load/store*
12	1	0	1	0	SP	Rd			Word8								*Load address*
13	1	0	1	1	0	0	0	0	S	SWord7							*Add offset to stack pointer*
14	1	0	1	1	L	1	0	R	Rlist								*Push/pop registers*
15	1	1	0	0	L	Rb			Rlist								*Multiple load/store*
16	1	1	0	1	Cond				Soffset8								*Conditional branch*
17	1	1	0	1	1	1	1	1	Value8								*Software Interrupt*
18	1	1	1	0	0	Offset11											*Unconditional branch*
19	1	1	1	1	H	Offset											*Long branch with link*
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

8086 – Instruction Word Size



RISC vs CISC – Instruction Complexity

- Instruction Complexity
  - **RISC**
    - Simple task
    - Example: Add, Subtract
  - **CISC**
    - Complex tasks
    - Example: Divide, Modulo

ARM – Instruction Complexity

Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)

8086 – Instruction Complexity

Mnemonic	Meaning	Condition	Mnemonic	Meaning	Condition
JA	above	CF = 0 and ZF = 0	JAE	above or equal	CF = 0
JB	below	CF = 1	JBE	below or equal	CF = 1 or ZF = 1
JC	carry	CF = 1	JCXZ	CX register is zero	(CF or ZF) = 0
JE	equal	ZF = 1	JG	greater	ZF = 0 and SF = OF
JGE	greater or equal	SF = OF	JL	less	(SF xor OF) = 1
JLE	less or equal	((SF xor OF) or ZF) = 1	JNA	not above	CF = 1 or ZF = 1
JNAE	not above nor equal	CF = 1	JNB	not below	CF = 0
JNBE	not below nor equal	CF = 0 and ZF = 0	JNC	not carry	CF = 0
JNE	not equal	ZF = 0	JNG	not greater	((SF xor OF) or ZF) = 1
JNGE	not greater nor equal	(SF xor OF) = 1	JNL	not less	SF = OF
JNLE	not less nor equal	ZF = 0 and SF = OF	JNO	not overflow	OF = 0
JNP	not parity	PF = 0	JNS	not sign	SF = 0
JNZ	not zero	ZF = 0	JO	overflow	OF = 1
JP	parity	PF = 1	JPE	parity even	PF = 1
JPO	parity odd	PF = 0	JS	sign	SF = 1
JZ	zero	ZF = 1			

RISC vs CISC – Number of Instructions

- Number of Instructions
  - **RISC**
    - Fewer instructions in instruction set
    - Example: ARM has 57
  - **CISC**
    - Many instructions in instruction set
    - Example: x86 has 981

ARM Instructions (RISC)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			*Move shifted register*
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			*Add/subtract*
3	0	0	1	Op		Rd			Offset8								*Move/compare/add /subtract immediate*
4	0	1	0	0	0	0	Op				Rs			Rd			*ALU operations*
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			*Hi register operations /branch exchange*
6	0	1	0	0	1	Rd			Word8								*PC-relative load*
7	0	1	0	1	L	B	0	Ro			Rb			Rd			*Load/store with register offset*
8	0	1	0	1	H	S	1	Ro			Rb			Rd			*Load/store sign-extended byte/halfword*
9	0	1	1	B	L	Offset5					Rb			Rd			*Load/store with immediate offset*
10	1	0	0	0	L	Offset5					Rb			Rd			*Load/store halfword*
11	1	0	0	1	L	Rd			Word8								*SP-relative load/store*
12	1	0	1	0	SP	Rd			Word8								*Load address*
13	1	0	1	1	0	0	0	0	S	SWord7							*Add offset to stack pointer*
14	1	0	1	1	L	1	0	R	Rlist								*Push/pop registers*
15	1	1	0	0	L	Rb			Rlist								*Multiple load/store*
16	1	1	0	1	Cond				Soffset8								*Conditional branch*
17	1	1	0	1	1	1	1	1	Value8								*Software Interrupt*
18	1	1	1	0	0	Offset11											*Unconditional branch*
19	1	1	1	1	H	Offset											*Long branch with link*
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

- **Small number of simple instructions**

8086 Instructions (CISC)

8086 INSTRUCTION SET

OPCODE		DESCRIPTION
AAA		ASCII adjust addition
AAD		ASCII adjust division
AAM		ASCII adjust multiply
AAS		ASCII adjust subtraction
ADC	dt,sc	Add with carry
ADD	dt,sc	Add
AND	dt,sc	Logical AND
CALL	proc	Call a procedure
CBW		Convert byte to word
CLC		Clear carry flag
CDL		Clear direction flag
CLI		Clear interrupt flag
CMC		Complement carry flag
CMP	dt,sc	Compare
CMPS	[dt,sc]	Compare string
CMPSB	"	" bytes
CMPSW	"	" words
CWD		Convert word to double word
DAA		Decimal adjust addition
DAS		Decimal adjust subtraction
DEC	dt	Decrement
DIV	sc	Unsigned divide
ESC	code,sc	Escape
HLT		Halt
IDIV	sc	Integer divide
IMUL	sc	Integer multiply
IN	ac,port	Input from port
INC	dt	Increment
INT	type	Interrupt
INTO		Interrupt if overflow
IRET		Return from interrupt
JA	slabel	Jump if above
JAE	slabel	Jump if above or equal
JB	slabel	Jump if below
JBE	slabel	Jump if below or equal
JC	slabel	Jump if carry
JCXZ	slabel	Jump if CX is zero
JE	slabel	Jump if equal
JG	slabel	Jump if greater
JGE	slabel	Jump if greater or equal
JL	slabel	Jump if less
JLE	slabel	Jump if less or equal
JMP	label	Jump
JNA	slabel	Jump if not above
JNAE	slabel	Jump if not above or equal
JNB	slabel	Jump if not below
JNBE	slabel	Jump if below or equal
JNC	slabel	Jump if no carry
JNE	slabel	Jump if not equal
JNG	slabel	Jump if not greater
JNGE	slabel	Jump if not greater or equal
JNL	slabel	Jump if not less
JNLE	slabel	Jump if not less or equal
JNZ	slabel	Jump if not zero
JNO	slabel	Jump if not overflow
JNP	slabel	Jump if not parity
JNS	slabel	Jump if not sign
JO	slabel	Jump if overflow
JPO	slabel	Jump if parity odd
JP	slabel	Jump if parity
JPE	slabel	Jump if parity even
JS	slabel	Jump if sign
JZ	slabel	Jump if zero
LAHF		Load AH from flags
LDS	dt,sc	Load pointer using DS
LEA	dt,sc	Load effective address
LES	dt,sc	Load pointer using ES
LOCK		Lock bus
LODS	[sc]	Load string
LODSB	"	" bytes
LODSW	"	" words
LOOP	slabel	Loop
LOOPE	slabel	Loop if equal
LOOPZ	slabel	Loop if zero
LOOPNE	slabel	Loop if not equal
LOOPNZ	slabel	Loop if not zero
MOV	dt,sc	Move
MOVS	[dt,sc]	Move string
MOVSB	"	" bytes
MOVSW	"	" words
MUL	sc	Unsigned multiply
NEG	dt	Negate
NOP		No operation
NOT	dt	Logical NOT
OR	dt,sc	Logical OR
OUT	port,ac	output to port
POP	dt	Pop word off stack
POPF		Pop flags off stack
PUSH	sc	Push word onto stack
PUSHF		Push flags onto stack
RCL	dt,cnt	Rotate left through carry
RCR	dt,cnt	Rotate right through carry
REP		Repeat string operation
REPE		Repeat while equal
REPZ		Repeat while zero
REPNE		Repeat while not equal
REPNZ		Repeat while not zero
RET	[pop]	Return from procedure
ROL	dt,cnt	Rotate left
ROR	dt,cnt	Rotate right
SAHF		Store AH into flags
SAL	dt,cnt	Shift arithmetic left
SHL	dt,cnt	Shift logical left
SAR	dt,cnt	Shift arithmetic right
SBB	dt,sc	Subtract with borrow
SCAS	[dt]	Scan string
SCASB	"	" byte
SCASW	"	" word
SHR	dt,cnt	Shift logical right
STC		Set carry flag
STD		Set direction flag
STI		Set interrupt flag
STOS	[dt]	Store string
STOSB	"	" byte
STOSW	"	" word
SUB	dt,sc	Subtraction
TEST	dt,sc	Test (logical AND)
WAIT		Wait for 8087
XCHG	dt,sc	Exchange
XLAT	table	Translate
XLATB	"	"
XOR	dt,sc	Logical exclusive OR

Notes:
dt - destination
sc - source
label - may be near or far address
slabel - near address

- Large number of complex instructions

http://mayurkalablogs.blogspot.com/2012/02/8086-instruction-set.html

RISC vs CISC Summary

- RISC : Simple Instructions
  1. simple addressing : EA
  2. most instructions fit in a single word
  3. smaller number of simple instructions in ISA
  4. arithmetic/logic operations on registers only
  5. load/store architecture for data transfers
- CISC : Complex Instructions
  1. more complex addressing : EA
  2. instructions spanning one or more words
  3. larger number of complex instructions in ISA
  4. arithmetic/logic operations on memory/register
  5. memory-to-memory data transfers