I. Architecture
A. 10 16-bit Registers
   1. 4 General Purpose, AX, BX, CX, DX
      a) Each accessible as 2 8-bit registers, AH, AL, BH, BL, CH, CL, DH, DL
      b) BX can be used like CUSP FP
   2. 4 Specialized Registers
      a) SP = Stack Pointer
      b) BP = Base Pointer, like CUSP FP
      c) SI = Source Index, like CUSP XR
      d) DI = Destination Index, like CUSP XR
   3. 4 Segment Registers
      a) CS = Code Segment base address.
      b) DS = Data Segment base address.
      c) SS = Stack Segment base address.
      d) ES = Extra Segment base address.
   4. IP = Instruction Pointer, like CUSP PC
   5. Flags including sign flag (CUSP LT); zero flag (CUSP EQ); interrupt flag (CUSP IE); signed overflow and
      unsigned overflow flags (CUSP OV); and 4 other flags.
B. Memory, 1 MByte $00000 - $FFFFF
C. I/O Ports 64 Kbyte, $0000 - $FFFFF
D. Bytes and Words
   1. Bytes $00000 and $00001 are Word $00000, Bytes $00001 and $00002 are Word $00001
   2. Separate operations to manipulate bytes and words.

II. Memory Access
A. Addressing Modes
   1. All transfers have same mnemonic, MOV. MOV destination, source
   2. Operands determine which opcode. This is done invisibly by the assembler.
      a) When a given operation can be done on a byte or a word the opcodes are the same except the least significant
         bit is a zero for 8-bit and a one for 16-bit.
      b) Possible operands: segment register, 8 bit register, 16 bit general purpose register, address, numerical value.
      c) There are sometimes specialized opcodes that overlap more general opcodes, but allow shorter instructions.
   3. Elements in address determine addressing mode.
      a) Direct: addr. Use DS:# to indicate direct mode instead of immediate mode when working with numerical
         values.
         (1) Label type, numerical Vs. address, dependent on how it is defined.
      b) Frame addressing uses either BX or BP.
      c) Index addressing uses either DI or SI.
      d) Immediate mode determined by whether value is explicitly or implicitly numerical. Labels that are equated to
         numerical values cause immediate mode.
      e) No indirect addressing.
         (1) It is accomplished using "register indirect addressing" in two steps. 1.) Load the base address into a base
         register (BX or BP), or an index register (SI, DI). 2.) Load using the base register and zero displacement
   4. Relative Addressing
      a) Jumps use an 8-bit or 16-bit displacement following the opcode to determine new IP. 8-bit allows 128
         backward and 127 forward.
      b) Permits smaller code.
      c) Position independent so it allows relocatable code.
   5. Instruction Families
      a) Move: MOV has
         | Dest | Src | Examples | Opcode |
         |------|-----|----------|--------|
         | reg8 | r/m8; |          | $8A    |
         | reg16| r/m16; |         | $8B    |
         | r/m8 | imm8; |          | $C6    |
         | r/m16| imm16; |        | $C7    |
b) Math and compare: ADD, SUB, CMP, AND, OR, XOR have r/m16,imm8 addressing mode, but no modes for segment registers.

c) Single operand: NEG, NOT, PUSH, POP, INC, DEC work with r/m8, r/m16, reg16 using $xx + reg code (not NOT or NEG), sreg (PUSH and POP only).

d) Jumps: Conditional jumps use "relative addressing" by adding a signed 8 bit displacement to IP based on the operand. An unconditional jump to anywhere in the IP's 64Kbyte range using a signed 16-bit displacement.

e) Non-operand: e.g. CLI (clear IF), PUSHF, POPF, and NOP.

6. Relative addressing permits relocatable object code.

B. Instruction Encoding

1. May be one to five bytes long. Variable length instructions allow more efficient use of space, but will slow the performance.

2. First byte is opcode.


   a) mod (bits 7-6). 00 = no displacement, 01 = 8 bit displacement, 10 = 16 bit displacement, 11 = treat r/m as a register.

   b) reg (bits 5-3). If 8-bit opcode then 000 = AL, 001 = CL, 010 = DL, 011 = BL, 100 = AH, 101 = CH, 110 = DH, 111 = BH. If 16-bit opcode then 000 = AX, 001 = CX, 010 = DX, 011 = BX, 100 = SP, 101 = BP, 110 = SI, 111 = DI.

   (1) DEC and INC share the same opcodes. $FE (8 bit) and $FF(16 bit), reg = 000 means INC, reg = 001 means DEC.

   (2) For MOV using sreg, whether src or dest, reg = 02 sreg, where sreg ES = 002, CS = 012, SS =102, DS =112

   C) r/m (bits 2-0). If mod is 11 then same encoding as reg, else 000 = BX + SI + disp, 001 = BX + DI + disp, 010 = BP + SI + disp, 011 = BP + DI + disp, 100 = SI + disp, 101 = DI + disp, 110 = BP + disp except when mod is 00 then just disp, 111 = BX + disp.

4. Later bytes hold the displacement:

   a) None when mod is 11

   b) One byte when mod is 01. Displacement is calculated using sign extension.

   c) Two bytes when mod is 00 with r/m 110 or mod is 10, with LSB in low address, e.g. MOV BX, DS:4567h becomes $8B1E6745.

5. Some opcodes have the register and mode built-in so second byte format is not used. For example MOV AL, mem8 is encoded as $A0 low_address high_address.

C. Segmentation

1. After calculation of the 16-bit address of instruction, with carries ignored, the address is added to the value of a segment register which has been shifted to the left 4 bits. This produces a 20 bit address.

2. Segment register to use is based on the type of memory access involved. CS for instruction fetches; SS for PUSH, POP, and instructions using BP; DS for data. These standards can be overridden with a prefix byte.

3. 32-bit far pointers are used to access inter-segment variables. The first word is the offset, and the second word is the value to be stored in a segment register. Use LDS reg16, mem32 to load the DS with segment address and reg16 with offset.

4. Intersegment control transfer instructions have opcode followed by 16-bit IP and 16-bit CS addresses. There are Intersegment jumps, calls, and returns.

5. In protected mode the segment registers are indices into a segmentation description table of 24-bit addresses that are added to the instruction's address.

III. Procedures

   A. Instead of caller taking care of PUSHed parameters the RET n instruction first POPs the IP and then adds n to the SP.
B. Variable parameters are expected to be far pointers.
C. All code should be relocatable, i.e., assume CS is $0000

IV. String Operations
A. MOVSB (Move String Byte) moves a byte of memory from DS:[SI] to ES:[DI], and then either increments or decrements SI and DI depending on the DF (direction flag). If DF = 1 then increment else decrement. CLD clears the direction flag, and STD sets the direction flag.
B. REP prefix causes the following string operation to be done the number of times indicated in the CX. The CX is decremented prior to each instruction and is left containing 0.
C. STOSB (Store String Byte) copies the AL into ES:[DI], and updates DI. Can be used with REP to fill bytes with the same value.
D. CMPSB (Compare String Byte) compares the two bytes at DS:[SI] and ES:[DI], updates SI and DI, and sets flags.
E. SCASB (Scan String Byte) compares the ES:[DI] and with AL, updates DI, and sets flags.
F. REPE (Repeat while equal) prefix causes the following string operation to continue until the ZF (zero flag) is cleared.
G. REPNE (Repeat while not equal) prefix causes the following string operation to continue until the ZF is set.
H. LODSB (Load string byte) loads AL with ES:[DI] and updates DI.
I. MOVSW, STOSW, CMPSW, SCASW do the same thing as their byte equivalents except they act on words and change the SI and DI registers by two instead of one.

V. Jumps
A. JA and JB for unsigned comparisons
B. JG and JL for signed comparisons.
C. JCXZ is specialized to compare CX with zero because CX is used as a counter for REP.

VI. Input/Output
A. All data transferred to and from the AL (8-bit operations) or AX (16-bit operations) using IN or OUT. Since I/O direct addressing is only 8-bit, you must use register indirect addressing with DX holding the port address to access ports beyond first 256.
B. Effects of reading or writing to a particular I/O port is dependent on the other components attached to the CPU, and not the CPU itself.
   1. For writing to the printer, the character is stored in the data port, then set and clear the STROBE bit in the control port.
C. Interrupts
   1. Uses an interrupt controller that can handle many devices, and passes the interrupts one at a time to the CPU.
   2. Controller provides a interrupt vector number (0 - 255) to the CPU.
   3. When the IE flag is set and an interrupt occurs the CPU does three things:
      a) 1) Pushes the flags, CS, and IP onto the stack.
      b) 2) Clears IE.
      c) 3) Loads CS and IP with contents of the interrupt vector which is the four memory bytes starting at location vector number * 4.
   4. Before returning from an Interrupt Service Routine (ISR), the routine must inform the interrupt controller which interrupt has been handled.
   5. Programs can control priorities, mask (ignore), unmask (accept), and poll the interrupt requests from devices.
   6. Programs can substitute their own ISRs for system ISRs. Care must be taken to ensure that critical system ISRs are called from the new ISRs.

VII. Arithmetic
A. Has separate signed and unsigned flags that are always affected.
B. Multiples
   1. Multiplicand in AL or AX.
   2. Separate signed and unsigned multiplication instructions.
   3. 8-bit stored in AH and AL. 16-bit stored in DX and AX.
C. Division
   1. Number to be divided is in AL or AX.
   2. Separate signed and unsigned divisions.
   3. 8-bit stores quotient in AL and remainder in AH. 16-bit stores quotient in AX and remainder in DX.
D. Multiple Precision Arithmetic
   1. ADC (Add with carry) allows addition of multi-word integers. By initially clearing the CF and then adding from LSW to MSW and including the Carry bit as a LSB in each addition, the 8086 can add any size integers.
2. SBB (Subtract with borrow) allows subtraction of multi-word integers. By initially clearing the CF and then subtracting from LSW to MSW and subtracting the Carry bit as a LSB in each subtracting, the 8086 can subtract any size integers.

E. Floating Point Operations.
   1. 8086 has no floating point instructions. The 8087 coprocessor chip provides the functionality.
   2. 8087 has a stack of eight 80-bit floating point accumulators. The 8087 also has a field in its status register that is 3-bit stack pointer.
   3. Four step process:
      a) 1) Push a value onto the 8087 stack (FILD or FLD). The value is automatically converted to an 80-bit floating point value.
      b) 2) Issue an arithmetic command with second value to the 8087 (FIMUL, FMUL, FIDIV, FDIV etc.).
      c) 3) Wait for the coprocessor (FWAIT).
      d) 4) Pop the result from the stack (FSTP)

VIII. Sample Chapter 11 questions.
   A. What would be the consequences of allowing a timer interrupt to occur in between the two instructions required to load an interrupt vector?
   B. Questions like 11-2, 11-3, and 11-4.
   C. The 8086 has four segment registers to permit access to memory beyond the 64K limit a 2-byte word displacement permits.
      1. Describe the characteristics of a program that accesses memory inefficiently. Think about each segment register and how a program uses it.
      2. How would you rewrite a program if there were only one segment register? Again, think about how each register is used.
   D. What are the advantages of relative addressing?
   E. Unlike CUSP, the 8086 does not have predefined I/O ports for specific devices. What are the advantages and disadvantages of this for a programmer?
   F. What aspects[s] of programming are facilitated by the 8087 having a stack of floating point accumulators, instead of just one?