INTRODUCTION TO MICROPROCESSOR 8086

The **8086** is a 16- bit microprocessor chip designed by Intel in 1978, which gives rise to the x86 architecture. Intel 8088, released in 1979, was essentially the same chip, but with an external 8-bit data bus (allowing the use of cheaper and fewer supporting logic chips), and is notable as the processor used in the original IBM PC.

BACKGROUND

The 8086 was intended as a temporary substitute for the ambitious iAPX 432 project in an attempt to draw attention from other manufacturers (such as Motorola, Zilog, and National Semiconductor) less delayed 16 and 32-bit processors. Both the architecture and the physical chip were therefore developed very quickly, and were based on the earlier 8080 and 8085 designs with a similar register set. The chip had around 29,000 transistors (many for microcode) and would also function as a continuation of the 8085; although not directly source code compatible, it was designed so that assembly language for the 8085 could be automatically converted into (sub-optimal) 8086 assembly source, usually with little or no hand-editing. However, the 8086 design was expanded to support full 16 –bit processing instead of the fairly basic 16-bit capabilities of the Intel 8080/8085.

BUSES AND OPERATION

Address Bus- 20-bit address bus. Can access 2^{20} memory locations i.e. 1MB of memory.

Data Bus- 16-bit data bus. Can access 16-bit data in one operation. (All registers of the 8086 are 16-bits wide, further contributing to the moniker of "16-bit microprocessor").

Control Buses- Carries the essential signals for various operations.

8086 instructions varied from 1 to 6 bytes. Therefore, fetch and execution were (and still are) concurrent. The bus interface unit feeds the instruction stream to the execution unit through a 6 byte fetch queue, a form of loosely coupled pipelining.

REGISTERS AND INSTRUCTIONS

The processor featured eight 16-bit registers including the stack pointer. Four of them could also be accessed as eight 8-bit registers.

Due to a compact encoding inspired by 8085 and other 8-bit processors, most instructions were *one- or two address* operations which means that the result were stored in one of the operands. A single memory location could also often be used as both source and destination operand which, among other factors, could further contribute to a rather small executable code foot-print.

Although the degree of orthogonality between registers and operations were greater than in 8085, it is still low, and data registers were also sometimes used implicitly by instructions. While perfectly sensible for the assembly programmer, this complicates register allocation for compilers.

8086 also featured 64K 8-bit I/O ports (or 32K 16 bit), and fixed vectored interrupts.

SEGMENTATION

There were also four segment registers that could be set from index registers. The segment registers allowed the CPU to access one megabyte + 64 KB – 16 bytes of memory in an odd way. Rather than just supplying missing bytes, as in most segmented processors, the 8086 shifted the segmented register left 4 bits and added it to the offset address, thus:

$$\textit{Physical address} = \textit{segment} \times 16 + \textit{offset}$$

The physical memory address was therefore 20 bits wide (while both segment and offset were 16 bits). As a result of this scheme, segments overlapped, making it possible to have up to 4096 *different* pointers addressing the same location. While acceptable, and even useful, for assembly language programming (where control of the segments was complete) it caused confusion and was considered poor design by most people, forcing entirely new concepts (**near** and **far** keywords) into languages such as Pascal and C.

Although this scheme made expanding the address space to more than 2^{20} bytes more difficult, it was nevertheless soon expanded by a new MMU- controlled addressing scheme in the 80286's protected mode. Later, and on top of this, 80386expanded the whole general purpose registers set (and hence, the offsets) to 32 bits, thereby enabling a *linear* addressing range of 2^{32} bytes (with a *total* range of 2^{36}). However, those chips were, for a long period of time, often used in real mode, remaining compatible with older OS's.

In a similar way, early programs could ignore the segments, and just use plain 16-bit addressing, which allowed 8-bit software to be easily ported to the 8086. The authors of MS-DOS took advantage of this by providing an API very similar to CP/M. This was important when the 8086 was new, because it allowed many existing CP/M applications to be quickly made available on the new platform, which greatly eased the transition.

INTEL 8086 MICROPROCESSOR ARCHITECTURE

MEMORY

Program, data and stack memories occupy the same memory space. The total addressable memory size is 1MB KB. As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory. To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory (see the "Registers" section below).

16-bit pointers and data are stored as:
address: low-order byte
address+1: high-order byte

32-bit addresses are stored in "segment: offset" format as:
address: low-order byte of segment
address+1: high-order byte of segment
address+2: low-order byte of offset
address+3: high-order byte of offset

Physical memory address pointed by segment: offset pair is calculated as:

address = (* 16) + <offset>

Program memory - program can be located anywhere in memory. Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory. All conditional jump instructions can be used to jump within approximately +127 - -127 bytes from current instruction.

Data memory - the processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB (if all four segments point to different 64 KB blocks). Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES: (some registers and instructions by default may use the ES or SS segments instead of DS segment).

Word data can be located at odd or even byte boundaries. The processor uses two memory accesses to read 16-bit word located at odd byte boundaries. Reading word data from even byte boundaries requires only one memory access.

Stack memory can be placed anywhere in memory. The stack can be located at odd memory addresses, but it is not recommended for performance reasons (see "Data Memory" above).

Reserved locations:

- 0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment: offset.
- FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.

INTERRUPTS

The processor has the following interrupts:

INTR is a mask able hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction. When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

NMI is a non-mask able interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the mask able interrupt.

Software interrupts can be caused by:

- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: divide error (type 0), unused opcode (type 6) and escape opcode (type 7).

Software interrupt processing is the same as for the hardware interrupts.

I/O PORTS

65536 8-bit I/O ports. These ports can be also addressed as 32768 16-bit I/O ports.

REGISTERS

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

Accumulator register consists of 2 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

Base register consists of 2 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of 2 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

Data register consists of 2 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

The following registers are both general and index registers:

Stack Pointer (SP) is a 16-bit register pointing to program stack.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

Instruction Pointer (IP) is a 16-bit register.

Flags is a 16-bit register containing 9 1-bit flags:

- Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- Direction Flag (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.
- Single-step Flag (TF) - if set then single-step interrupt will occur after the next instruction.
- Sign Flag (SF) - set if the most significant bit of the result is set.
- Zero Flag (ZF) - set if the result is zero.
- Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

INSTRUCTION SET

8086 instruction set consists of the following instructions:

- Data moving instructions.
- Arithmetic - add, subtract, increment, decrement, convert byte/word and compare.
- Logic - AND, OR, exclusive OR, shift/rotate and test.
- String manipulation - load, store, move, compare and scan for byte/word.
- Control transfer - conditional, unconditional, call subroutine and return from subroutine.
- Input/Output instructions.
- Other - setting/clearing flag bits, stack operations, software interrupts, etc.

ADDRESSING MODES

Implied - the data value/data address is implicitly associated with the instruction.

Register - references the data in a register or in a register pair.

Immediate - the data is provided in the instruction.

Direct - the instruction operand specifies the memory address where data is located.

Register indirect - instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.

Based - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.

Indexed - 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

Based Indexed - the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

Based Indexed with displacement - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

	CMPSB				MOV		
AAA	CMPSW	JAE	JNBE	JPO	MOVSB	RCR	SCASB
AAD	CWD	JB	JNC	JS	MOVSW	REP	SCASW
AAM	DAA	JBE	JNE	JZ	MUL	REPE	SHL
AAS	DAS	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DEC	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DIV	JE	JNL	LEA	NOT	REPZ	STD
AND	HLT	JG	JNLE	LES	OR	RET	STI
CALL	IDIV	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IMUL	JL	JNP	LODSW	POP	ROL	STOSW
CLC	IN	JLE	JNS	LOOP	POPA	ROR	SUB
CLD	INC	JMP	JNZ	LOOPE	POPF	SAHF	TEST
CLI	INT	JNA	JO	LOOPNE	PUSH	SAL	XCHG
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA	SAR	XLATB
CMP	IRET	JNB	JPE	LOOPZ	PUSHF	SBB	XOR
	JA				RCL		

Complete 8086 Instruction Set

8086/80186/80286/80386/80486 Instruction Set:

AAA - Ascii Adjust for Addition
AAD - Ascii Adjust for Division
AAM - Ascii Adjust for Multiplication
AAS - Ascii Adjust for Subtraction
ADC - Add With Carry
ADD - Arithmetic Addition
AND - Logical And
ARPL - Adjusted Requested Privilege Level of Selector (286+ PM)
BOUND - Array Index Bound Check (80188+)
BSF - Bit Scan Forward (386+)
BSR - Bit Scan Reverse (386+)
BSWAP - Byte Swap (486+)
BT - Bit Test (386+)
BTC - Bit Test with Compliment (386+)
BTR - Bit Test with Reset (386+)
BTS - Bit Test and Set (386+)
CALL - Procedure Call
CBW - Convert Byte to Word
CDQ - Convert Double to Quad (386+)
CLC - Clear Carry
CLD - Clear Direction Flag
CLI - Clear Interrupt Flag (disable)
CLTS - Clear Task Switched Flag (286+ privileged)
CMC - Complement Carry Flag
CMP - Compare
CMPS - Compare String (Byte, Word or Doubleword)
CMPXCHG - Compare and Exchange
CWD - Convert Word to Doubleword
CWDE - Convert Word to Extended Doubleword (386+)
DAA - Decimal Adjust for Addition

DAS - Decimal Adjust for Subtraction
DEC - Decrement
DIV - Divide
ENTER - Make Stack Frame (80188+)
ESC - Escape
HLT - Halt CPU
IDIV - Signed Integer Division
IMUL - Signed Multiply
IN - Input Byte or Word From Port
INC - Increment
INS - Input String from Port (80188+)
INT - Interrupt
INTO - Interrupt on Overflow
INVD - Invalidate Cache (486+)
INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)
IRET/IRETD - Interrupt Return
Jxx - Jump Instructions Table
JCXZ/JECXZ - Jump if Register (E) CX is Zero
JMP - Unconditional Jump
LAHF - Load Register AH From Flags
LAR - Load Access Rights (286+ protected)
LDS - Load Pointer Using DS
LEA - Load Effective Address
LEAVE - Restore Stack for Procedure Exit (80188+)
LES - Load Pointer Using ES
LFS - Load Pointer Using FS (386+)
LGDT - Load Global Descriptor Table (286+ privileged)
LIDT - Load Interrupt Descriptor Table (286+ privileged)
LGS - Load Pointer Using GS (386+)
LLDT - Load Local Descriptor Table (286+ privileged)
LMSW - Load Machine Status Word (286+ privileged)
LOCK - Lock Bus
LODS - Load String (Byte, Word or Double)
LOOP - Decrement CX and Loop if CX Not Zero
LOOPE/LOOPZ - Loop While Equal / Loop While Zero
LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal
LSL - Load Segment Limit (286+ protected)
LSS - Load Pointer Using SS (386+)
LTR - Load Task Register (286+ privileged)
MOV - Move Byte or Word
MOVS - Move String (Byte or Word)
MOVSX - Move with Sign Extend (386+)
MOVZX - Move with Zero Extend (386+)
MUL - Unsigned Multiply
NEG - Two's Complement Negation
NOP - No Operation (90h)
NOT - One's Compliment Negation (Logical NOT)
OR - Inclusive Logical OR
OUT - Output Data to Port
OUTS - Output String to Port (80188+)
POP - Pop Word off Stack

POPA/POPAD - Pop All Registers onto Stack (80188+)
POPF/POPFD - Pop Flags off Stack
PUSH - Push Word onto Stack
PUSHA/PUSHAD - Push All Registers onto Stack (80188+)
PUSHF/PUSHFD - Push Flags onto Stack
RCL - Rotate Through Carry Left
RCR - Rotate Through Carry Right
REP - Repeat String Operation
REPE/REPZ - Repeat Equal / Repeat Zero
REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero
RET/RETF - Return From Procedure
ROL - Rotate Left
ROR - Rotate Right
SAHF - Store AH Register into FLAGS
SAL/SHL - Shift Arithmetic Left / Shift Logical Left
SAR - Shift Arithmetic Right
SBB - Subtract with Borrow/Carry
SCAS - Scan String (Byte, Word or Doubleword)
SETAE/SETNB - Set if Above or Equal / Set if Not Below (386+)
SETB/SETNAE - Set if Below / Set if Not Above or Equal (386+)
SETBE/SETNA - Set if Below or Equal / Set if Not Above (386+)
SETE/SETZ - Set if Equal / Set if Zero (386+)
SETNE/SETNZ - Set if Not Equal / Set if Not Zero (386+)
SETL/SETNGE - Set if Less / Set if Not Greater or Equal (386+)
SETGE/SETNL - Set if Greater or Equal / Set if Not Less (386+)
SETLE/SETNG - Set if Less or Equal / Set if Not greater or Equal(386+)
SETG/SETNLE - Set if Greater / Set if Not Less or Equal (386+)
SETS - Set if Signed (386+)
SETNS - Set if Not Signed (386+)
SETC - Set if Carry (386+)
SETNC - Set if Not Carry (386+)
SETO - Set if Overflow (386+)
SETNO - Set if Not Overflow (386+)
SETP/SETPE - Set if Parity / Set if Parity Even (386+)
SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)
SGDT - Store Global Descriptor Table (286+ privileged)
SIDT - Store Interrupt Descriptor Table (286+ privileged)
SHL - Shift Logical Left
SHR - Shift Logical Right
SHLD/SHRD - Double Precision Shift (386+)
SLDT - Store Local Descriptor Table (286+ privileged)
SMSW - Store Machine Status Word (286+ privileged)
STC - Set Carry
STD - Set Direction Flag
STI - Set Interrupt Flag (Enable Interrupts)
STOS - Store String (Byte, Word or Doubleword)
STR - Store Task Register (286+ privileged)
SUB - Subtract
TEST - Test for Bit Pattern
VERR - Verify Read (286+ protected)
VERW - Verify Write (286+ protected)

WAIT/FWAIT - Event Wait
WBINVD - Write-Back and Invalidate Cache (486+)
XCHG - Exchange
XLAT/XLATB - Translate
XOR - Exclusive OR