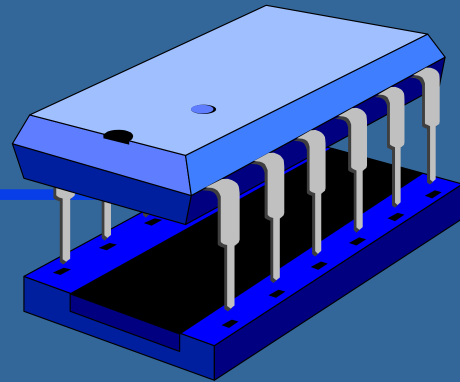

8086 Instructions

Dr Philip Leong
phwl@doc.ic.ac.uk

Topics

- Expressions
- Overflow and Divide by Zero
- Booleans & Comparison
- If Statements & Loops

8086 Instruction Set (79 basic instructions)

- AAA AAD AAM AAS ADC ADD AND
- CALL CBW CLC CLD CLI CMC CMP CMPS CMPXCHG
- CWD DAA DAS DEC DIV ESC HLT IDIV
- IMUL IN INC INT INTO IRET/IRETD
- Jxx JCXZ/JECXZ JMP
- LAHF LDS LEA LES LOCK LODS LOOP
- MOV MOVS LOOPE/LOOPZ LOOPNZ/LOOPNE
- MUL NEG NOP NOT OR OUT POP POPF/POPFD PUSH
- RCL RCR PUSHF/PUSHFD
- REP REPE/REPZ RET/RETF REPNE/REPNZ
- ROL ROR SAHF SAL/SHL SAR SBB SCAS SHL SHR
- STC STD STI STOS SUB
- TEST WAIT/FWAIT XCHG XLAT/XLATB XOR

Instruction types

Instructions vary from one CPU to another, General groupings possible:

- Arithmetic/Logic
  - Add, Subtract, AND, OR, shifts
  - Performed by ALU
- Data Movement
  - Load, Store (to/from registers/memory)
- Transfer of Control
  - Jump, Branch, procedure call
- Test/Compare
  - Set condition flags
- Input/Output
  - In, Out (Only on some CPUs)
- Others
  - Halt, NOP

Integer Addition & Subtraction

Instruction		Operation	Notes
ADD	dst, src	dst := dst + src	Addition
SUB	dst, src	dst := dst – src	Subtraction
CMP	dst, src	dst – src	Compare & Set FLAGS
INC	opr	opr := opr + 1	Increment by 1
DEC	opr	opr := opr – 1	Decrement by 1
NEG	opr	opr := – opr	Negate

- Operands can be byte, word or doubleword (dword) sized
- Arithmetic instructions also set Flag bits, e.g. the Zero Flag (ZF), the Sign Flag (SF), the Carry Flag (CF), the Overflow Flag (OF) which can be tested with branching instructions.

Integer Multiply & Divide

Instruction	Operation	Notes
IMUL opr	AX := AL * opr	Word = Byte * Byte
	DX:AX := AX * opr	Doubleword = Word * Word
	EDX:EAX := EAX * opr	Quadword = Dword* Dword
IDIV opr	AL := AX / opr AH := AX mod opr	Word / Byte
	AX := (DX:AX) / opr DX := (DX:AX) mod opr	Doubleword / Word
	EAX := (EDX:EAX) / opr EDX := (EDX:EAX) mod opr	Quadword / Doubleword

Operands must be registers or memory operands

Integer Multiply (Pentium)

Instruction	Operation
IMUL DestReg, SrcOpr	DestReg := DestReg * SrcOpr
IMUL DestReg, SrcReg, immediate	DestReg := SrcReg * immediate
IMUL DestReg, MemOpr, immediate	DestReg := MemOpr * immediate

Operands can be word or doubleword sized

More Instructions

Instruction	Operation	Notes
SAL dst, N	dst := dst * 2^N	Shift Arithmetic Left
SAR dst, N	dst := dst / 2^N	Shift Arithmetic Right

SAL/SAR are quick ways of multiplying/dividing by powers of 2. N must be a constant (immediate value) or the byte register CL.

CBW	AX := AL	Convert Byte to Word
CWD	DX:AX := AX	Convert Word to Doubleword
CWDE	EAX:= AX	Convert Word to Doubleword

CBW, CWD, CWDE extend a signed integer by filling the extra bits of destination with the sign bit of the operand (i.e. preserve value of result)

Expressions

- var alpha, beta, gamma : int (* global variables *)
  ...
  alpha := 7; beta := 4; gamma := –3;
  alpha := (alpha * beta + 5 * gamma) / (alpha – beta)
  ...

- In this example we will represent Integers as 16-bit 2's complement values and use direct addressing and data declaration directives for global variables:

```
alpha    DW    0
beta     DW    0
gamma    DW    0
```

Example

; alpha := 7; beta := 4; gamma := –3
; alpha := (alpha * beta + 5 * gamma) / (alpha – beta)

```
MOV     AX, alpha   ;   AX := alpha
IMUL    beta        ;   (DX:AX) := alpha * beta
MOV     BX, AX      ;   Save least sig. word in BX
MOV     AX, 5       ;   AX := 5
IMUL    gamma       ;   (DX:AX) := 5 * gamma
ADD     AX, BX      ;   AX := 5 * gamma + alpha * beta
```

	MOV	IMUL	MOV	MOV	IMUL	ADD
AX	0007	001C	001C	0005	FFF1	000D
BX			001C	001C	001C	001C
CX						
DX		0000	0000	0000	FFFF	FFFF

Regs shown in Hex

Example Continued

; alpha := 7; beta := 4; gamma := –3
; alpha := (alpha * beta + 5 * gamma) / (alpha – beta)

MOV	BX, alpha	;	BX := alpha
SUB	BX, beta	;	BX := alpha – beta
CWD		;	Sign extend AX to DX
IDIV	BX	;	AX := (DX:AX) / operand
		;	DX := (DX:AX) % operand
MOV	alpha, AX	;	alpha := *final value*

	Prev	MOV	SUB	CWD	IDIV	MOV
AX	000D	000D	000D	000D	0004	→ 0004
BX	001C	0007	0003	0003	0003	alpha
CX						
DX	FFFF	FFFF	FFFF	0000	0001	

Integer Overflow

- Most arithmetic operations can produce an overflow, for example for signed byte addition if

  $A + B > 127$ or if $A + B < -128$

- Instructions which result in an overflow set the Overflow Flag in the FLAGS register, which we can test, e.g.

  Overflow Test

```
          ADD   AL,  BL      ; Add, will set FLAGS.ZF if overflow
          JO    ov_label     ; Jump to ov_label  if Overflow
          ...
ov_label:                    ; Handle Overflow condition somehow?
```

Integer Divide by Zero

- Another erroneous condition is division by zero which causes an interrupt to occur (we will cover interrupts later in course).
- We can guard against this occurring by explicitly checking the divisor before division, e.g.

Divide by Zero Test

```
            CMP   BL,   0        ; Compare Divisor with Zero
            JE    zero_div       ; Jump if (Divisor) is Equal to zero
            ...                  ; Else perform division

zero_div:   ....                 ; Handle divide by zero somehow?
```

“LOGICAL” (Bit-level) Instructions

Instruction	Operation	Notes
AND dst, src	dst := dst & src	Bitwise AND
TEST dst, src	dst & src	Bitwise AND and set FLAGS
OR dst, src	dst := dst | src	Bitwise OR
XOR dst, src	dst := dst ^ src	Bitwise XOR
NOT opr	opr := ~ opr	Bitwise NOT

Typical Uses

AND is used to clear specific bits (given by 0 bits in src) in the dst.
OR is used to set specific bits (given by 1 bits in src) in the dst.
XOR is used to toggle/invert specific bits (given by 1 bits in src) in the dst.
TEST is used to test specific bit patterns.

Booleans

- We will use bytes to represent booleans with the following interpretation:

  False = 0, True = Non-Zero

- var man, rich, okay : boolean
  ...
  okay := (man AND rich) OR NOT man

```
MOV   AL, man     ;   AL := man
AND   AL, rich    ;   AL := man AND rich
MOV   AH, man     ;   AH := man
NOT   AH          ;   AH := NOT man
OR    AL, AH      ;   AL := (man AND rich) OR NOT man
MOV   okay, AL    ;   okay := AL
```

JUMP Instructions

Jump instructions take the form OPCODE label, e.g. JGE next

Opcode	Flag Conditions	Notes
JMP	Unconditional	Jump
JE or JZ	ZF = 1	Jump if Equal or Jump if Zero (=)
JNE or JNZ	ZF = 0	Jump if Not Equal or Jump if Not Zero

Signed Comparisons

JG	ZF = 0 and SF = 0	Jump if Greater than (>)
JGE	SF = 0	Jump if Greater than or Equal (>=)
JL	SF = 1	Jump if Less than (<)
JLE	ZF = 1 or SF = 1	Jump if Less than or Equal (<=)

More JUMP Instructions

Unsigned Comparisons

JA	ZF = 0 and CF = 0	Jump if Above (>)
JAE	CF = 0	Jump if Above or Equal (>=)
JB	CF = 1	Jump if Below (<)
JBE	ZF = 1 or CF = 1	Jump if Below or Equal (<=)

Miscellaneous

JO	OF = 1	Jump if Overflow, ditto for CF, SF & PF
JNO	OF = 0	Jump if No Overflow, ditto for ...
JCXZ	CX = 0	Jump if CX = 0
JECXZ	ECX = 0	Jump if ECX = 0

If Statement

If age<100 then
    statements
end if

```
if:     CMP   age, 100
        JL    stats
        JMP   endif
stats:
        ; statements
endif:
```

```
if:     CMP   age, 100
        JGE   endif
        ; statements
endif:
```

If (age >= 21) and (age <= 65) then
    statements
end if

```
if:     CMP   age, 21
        JL    endif
        CMP   age, 65
        JG    endif
        ; statements
endif:
```

IF-Then-Else Statement

If age < 100 then
    statements1
else
    statements2
end if

```
if:     CMP   age, 100
        JGE   else

        ; statements1
        JMP   endif

else:

        ; statements2

endif:
```

While Loop

```
loop
    exit when  age > 99
    statements
end loop
```

```
while:    CMP    age, 99
          JG     endwhile

          ; statements

          JMP    while

endwhile:
```

Repeat Loop

loop
    statements
    exit when age > 99
end loop

```
repeat:

        ; statements

        CMP     age, 99
        JLE     repeat

endrepeat:
```

For Loop

for age : 1 .. 99
    statements
end for

```
for:    MOV   age, 1

next:   CMP   age, 99
        JG    endfor

        ; statements

        INC   age
        JMP   next

endfor:
```

Think about

1. How high level language statements are translated to 8086 instructions
2. How integer multiplication could be implemented if the IMUL instruction didn't exist
3. How logical operations can be used to set and clear bits