EC325 Microprocessors
Integer Arithmetics

Yasser F. O. Mohammad

REMINDER 1: Statement Anatomy

- [name] [mnemonic] [operand(s)] [; comment]
- Zerocount: mov ecx,0 ; initialize counter

- Name
  - Ends with colon ':' for instructions but not directives

- Mnemonic
  - Indicates what the statement is about

- Operands
  - Optional and separated with comas

- Comment
  - Optional and starts with a semicolon

REMINDER 2: MOVs not allowed

- Source and destination in memory
- To and from FLAG register
- To and from IP
- Source and destination are segment registers
- Immediate to segment register
- Operands are not same size

REMINDER 3: XCHG

- xchg source1,source2
- Exchange the two values
- Equals 3 moves

Operand1	Operand2	Clock Cycles 386	486	Pentium	Number of Bytes	Opcode
register 8	register 8	3	3	3	2	86
register 8	memory byte	5	5	3	2–7	86
EAX/AX	register 32/16	3	3	2	1	
	ECX/CX					91
	EDX/DX					92
	EBX/BX					93
	ESP/SP					94
	EBP/BP					95
	ESI/SI					96
	EDI/DI					97
register 32/16	register 32/16	3	3	3	2	87
register 32/16	memory 32/16	5	5	3	2–7	87

Explicit Size Declaration

- The problem:
  - mov [ebx], 0

- The solution:
  - mov BYTE PTR [ebx],0
  - mov WORD PTR [ebx],0
  - mov DWORD PTR [ebx],0

Addition and Subtraction

- add destination, source
  - Dest=dest+source
- sub destination, source
  - Dest=dest-source
- inc operand
  - operand=operand+1
- dec operand
  - operand=operand-1
- neg operand
  - Operand=-operand (2's complement)
- Why 2's complement???
- Careful: SF does not mean sign if the inputs are unsigned

Example Additions and Subtractions

AX: 77 AC CX: 4B 35	`add ax, cx`	AX C2 E1 CX 4B 35 SF 1 ZF 0 CF 0 OF 1
EAX: 00 00 00 75 ECX: 00 00 01 A2	`sub ecx, eax`	EAX 00 00 00 75 ECX 00 00 01 2D SF 0 ZF 0 CF 0 OF 0
BL: 4B	`add bl, 4`	BL 4F SF 0 ZF 0 CF 0 OF 0
DX: FF 20 word at value: FF 20	`sub dx, Value`	DX 00 00 Value FF 20 SF 0 ZF 1 CF 0 OF 0
EAX: 00 00 00 09	`add eax, 1`	EAX 00 00 00 0A SF 0 ZF 0 CF 0 OF 0
doubleword at Dbl: 00 00 01 00	`sub Dbl, 1`	Dbl 00 00 00 FF SF 0 ZF 0 CF 0 OF 0

Why do we use 2's Complement

- Arithmetic operations are the same in unsigned and 2's complement representations

BUT

- Flags mean different things
  - SF
    - Signed: sign
    - Unsigned: MSB
  - CF
    - Signed:
    - Unsigned: too large result
  - OF
    - Signed: too small or too large result
    - Unsigned:

Full Example

```
.DATA                       ; reserve storage for data
Prompt1     BYTE    "This program will evaluate the expression",cr,Lf,Lf
            BYTE    "     - (x + y - 2z + 1)",cr,Lf,Lf
            BYTE    "for your choice of integer values.",cr,Lf,Lf
            BYTE    "Enter value for x:  ",0
Prompt2     BYTE    "Enter value for y:  ",0
Prompt3     BYTE    "Enter value for z:  ",0
Value       BYTE    16 DUP (?)
Answer      BYTE    cr,Lf,"The result is "
Result      BYTE    6 DUP (?)
            BYTE    cr,Lf,0

.CODE                               ; start of main program code
_start:
            output  Prompt1         ; prompt for x
            input   Value,16        ; read ASCII characters
            atoi    Value           ; convert to integer
            mov     dx,ax           ; x

            output  Prompt2         ; prompt for y
            input   Value,16        ; read ASCII characters
            atoi    Value           ; convert to integer
            add     dx,ax           ; x + y

            output  Prompt3         ; prompt for z
            input   Value,16        ; read ASCII characters
            atoi    Value           ; convert to integer
            add     ax,ax           ; 2*z
            sub     dx,ax           ; x + y - 2*z

            inc     dx              ; x + y - 2*z + 1
            neg     dx              ; - (x + y - 2*z + 1)

            itoa    Result,dx       ; convert to ASCII characters

            output  Answer          ; output label and result

            INVOKE  ExitProcess, 0  ; exit with return code 0

PUBLIC _start                       ; make entry point public
END                                 ; end of source code
```

Unsigned Multiplication

- mul operand
  - AX=AL*operand ; if byte
  - DX:AX=AX*operand ; if word
  - EDX:EAX=EAX*operand ; if dword

- CF, OF are set if the high order half is nonzero
- AF,SF,PF,ZF may be destroyed
- Unsigned multiplication

Before	*Instruction executed*	*After*
AX: 00 05 BX: 00 02 DX: ?? ??	`mul bx`	DX 00 00 AX 00 0A CF, OF 0
EAX: 00 00 00 0A EDX: ?? ?? ?? ??	`mul eax`	EDX 00 00 00 00 EAX 00 00 00 64 CF, OF 0

AX: ?? 05
byte at Factor: FF

`mul Factor`

AX 04 FB

CF, OF 1

Signed Multiplication IMUL

- imul source
  - AX=AL*operand ; if byte
  - DX:AX=AX*operand ; if word
  - EDX:EAX=EAX*operand ; if dword
  - CF, OF are set if the high order half is significant
- imul register, source
  - register=register*source
  - CF, OF are set if the result cannot fit into *register*
- imul register, source, immediate
  - register=source*immediate
  - CF, OF are set if the result cannot fit into *register*

Example IMUL

Before	*Instruction executed*	*After*
AX: 00 05 BX: 00 02 DX: ?? ??	`imul bx`	DX 00 00 AX 00 0A CF, OF 0
AX: ?? 05 byte at Factor: FF	`imul Factor`	AX FF FB CF, OF 0
EBX: 00 00 00 0A	imul ebx, 10	EBX 00 00 00 64 CF, OF 0
ECX: FF FF FF F4 doubleword at Double: FF FF FF B2	`imul ecx, Double`	ECX 00 00 03 A8 CF, OF 0
word at Value: 08 F2 BX: ?? ??	`imul bx,Value,1000`	BX F1 50 CF, OF 1

Area Calculation

```
.386
.MODEL FLAT

ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

INCLUDE io.h

cr          EQU    0dh   ; carriage return character
LF          EQU    0ah   ; linefeed character

.STACK  4096              ; reserve 4096-byte stack

.DATA                     ; reserve storage for data
prompt1     BYTE   "This program will find the area of a
                     rectangle",cr,Lf,Lf
            BYTE   "Width of rectangle?  ",0
prompt2     BYTE   "Length of rectangle?  ",0
value       BYTE   16 DUP (?)
answer      BYTE   cr,Lf,"The area of the rectangle is "
area        BYTE   11 DUP (?)
            BYTE   cr,Lf,0

.CODE                               ; start of main program code
_start:
Prompt:     output prompt1          ; prompt for width
            input  value,16         ; read ASCII characters
            atod   value            ; convert to integer
            mov    ebx,eax          ; width

            output prompt2          ; prompt for length
            input  value,16         ; read ASCII characters
            atod   value            ; convert to integer
            mul    ebx              ; length * width

            dtoa   area,eax         ; convert to ASCII characters
            output answer           ; output label and result

            INVOKE ExitProcess, 0   ; exit with return code 0
PUBLIC _start                       ; make entry point public
            END
```

Unsigned Division

- div source
  - Dividend = quotient*divisor+reminder
  - Inputs: [implicit] dividend, [explicit] divisor
  - Outputs: quotient and remainder
  - Divisor=source and cannot be immediate
  - My Destroy AF, CF, OF, PF, SF, ZF
  - Example:
    - div BYTE PTR [1000]

source (divisor) size	other operand (dividend)	quotient	remainder
byte	AX	AL	AH
word	DX:AX	AX	DX
doubleword	EDX:EAX	EAX	EDX

Signed Division

- idiv divisor
  - Same as idiv but quotient takes the sign of the operation
  - Sign of the remainder = sign of dividend
  - Sign of quotient is negative iff sign of dividend and divisor are different

Before	*Instruction executed*	*After*
EDX: 00 00 00 00 EAX: 00 00 00 64 EBX: 00 00 00 OD	`div ebx`	EDX 00 00 00 09 EAX 00 00 00 07
DX: 00 00 AX: 00 64 CX: 00 0D	`idiv cx`	DX 00 09 AX 00 07
AX: 00 64 byte at Divisor: 0D	`div Divisor`	AX 09 07

Division Errors

- Division by zero
- Insufficient space in quotient
  - Exampe: 2468A/2=12345 (cannot fit into AX)

Extension for division

- ; some instructions to calculate BX
- ; we want to divide BX by the word starting at [SI]
- MOV AX,BX ; move BX to AX
- DIV WORD PTR [SI] ; do division
- ; now quotient is in AX and remainder in DX

- *What is wrong?*

First Solution

- ; some instructions to calculate BX
- ; we want to divide BX by the word starting at [SI]
- MOV AX,BX ; move BX to AX
- **MOV DX,0**
- DIV WORD PTR [SI] ; do division
- ; now quotient is in AX and remainder in DX

- *What if BX was a signed number?*

Second solution

- ; some instructions to calculate BX
- ; we want to divide BX by the word starting at [SI]
- MOV AX,BX ; move BX to AX
- **MOV DX,oFFFFH**
- DIV WORD PTR [SI] ; do division
- ; now quotient is in AX and remainder in DX

- *What if BX was an unsigned or a positive number?*

Correct Solution

- ; some instructions to calculate BX
- ; we want to divide BX by the word starting at [SI]
- MOV AX,BX ; move BX to AX
- **CWD ; convert word to double word**
- IDIV WORD PTR [SI] ; do division
- ; now quotient is in AX and remainder in DX

Data Extension Instructions

- CBW
  - Sign extend AL into AX
- CWD
  - Sign extend AX into DX:AX
- CDQ
  - Sign extend EAX into EDX:EAX
- CWDE
  - Sign extend AX into EAX

Example Sign Extension

Before	*Instruction executed*	*After*				
AX: 07 0D DX: ?? ??	`cwd`	DX	00	00		
		AX	07	0D		
EAX: FF FF FA 13 EDX: ?? ?? ?? ??	`cdq`	EDX	FF	FF	FF	FF
		EAX	FF	FF	FF	13
AL: 53	`cbw`	AX	00	53		
AL: C6	`cbw`	AX	FF	C6		
AX: FF 2A	`cwde`	EAX	FF	FF	FF	2A

Moving with extension

- movzx Register, source
  - Moves the source to register and zero extends it
- movsx Register, source
  - Moves the source to the register and sign extends it

Before	*Instruction executed*	*After*				
word at value: 07 0D	`movsx ecx,value`	ECX	00	00	07	0D
word at Value: F7 0D	`movsx ecx,value`	ECX	FF	FF	F7	0D
word at Value: 07 0D	`movzx ecx,value`	ECX	00	00	07	0D
word at Value: F7 0D	`movzx ecx,value`	ECX	00	00	F7	0D

Full Program

```
.DATA                          ; reserve storage for data
Prompt1      BYTE   CR,LF,"This program will convert a Celsius "
             BYTE   "temperature to the Fahrenheit scale",cr,Lf,Lf
             BYTE   "Enter Celsius temperature:  ",0
Value        BYTE   10 DUP (?)
Answer       BYTE   CR,LF,"The temperature is"
Temperature  BYTE   6 DUP (?)
             BYTE   "   Fahrenheit",cr,Lf,0

.CODE                                ; start of main program code
_start:
Prompt:      output Prompt1          ; prompt for Celsius temperature
             input  Value,10         ; read ASCII characters
             atoi   Value            ; convert to integer

             imul   ax,9             ; C*9
             add    ax,2             ; rounding factor for division
             mov    bx,5             ; divisor
             cwd                     ; prepare for division
             idiv   bx               ; C*9/5
             add    ax,32            ; C*9/5 + 32

             itoa   Temperature,ax   ; convert to ASCII characters
             output Answer           ; output label and result

             INVOKE ExitProcess, 0   ; exit with return code 0
PUBLIC _start                        ; make entry point public
             END
```

ADC and SBB

- ADC dest, source
  - Dest=dest+source+CF
- SBB dest, source
  - Dest=dest-source-CF
- Used to add and subtract large numbers

Adding Large Numbers

```
Nbr1Hi   DWORD   ?      ; High order 32 bits of Nbr1
Nbr1Lo   DWORD   ?      ; Low order 32 bits of Nbr1
Nbr2Hi   DWORD   ?      ; High order 32 bits of Nbr2
Nbr2Lo   DWORD   ?      ; Low order 32 bits of Nbr2


mov   eax, Nbr1Lo      ; Low order 32 bits of Nbr1
add   eax, Nbr2Lo      ; add Low order 32 bits of Nbr2
mov   Nbr1Lo, eax      ; sum to destination
mov   eax, Nbr1Hi      ; High order 32 bits of Nbr1
adc   eax, Nbr2Hi     ; add High order 32 bits of Nbr2 & carry
mov   Nbr1Hi, eax      ; sum to destination
```

Carry Flag Control

Instruction	Operation	Clock Cycles	Number of Bytes	Opcode
`clc`	clear carry flag (CF := 0)	2	1	F8
`stc`	set carry flag (CF := 1)	2	1	F9
`cmc`	complement carry flag (if CF = 0 then CF := 1 else CF := 0)	2	1	F5