Assembly Cheat Sheet

Third Year Computer Science College Notes

Assembly - Intel

Author : Ahmed Ayman

Arithmetic Operations

```
; dest and src must be same size
; no memory to memory arithmetic
inc dest        ; increase dest by 1
dec dest        ; decrease dest by 1
add dest, src       ; add src on dest and store in dest
adc dest, src       ; dest = dest + src + CF
sub dest, src       ; subtract src from dest and store in dest
neg dest        ; negate dest by finding 2's complement of dest
; for mul, imul, div, and idiv no immediate values are permitted
mul src         ; unsigned multiplication
                ; multiplies 8, 16, or 32-bit operand by AL, AX, or
EAX and store it in AX, DX:AX, EDX:EAX
                ; The Carry and Overflow flags are set if upper
half of the product is non-zero
imul src        ; signed multiplication
                ; Preserves the sign of the product by sign-
extending it
                ; multiplies 8, 16, or 32-bit operand by AL, AX, or
EAX and store it in AX, DX:AX, EDX:EAX
                ; The Carry and Overflow flags are set if the upper
half of the product is not a sign extension of the lower half
div src         ; unsigned division
                ; divides AX, DX:AX, or EDX:EAX over 8, 16, or 32-
bit operand and store in AL, AX, or EAX the result and in AH, DX,
or EDX the reminder
idiv src        ; signed division
                ; Signed integers must be sign-extended before
division takes place.
                ; Fill high byte/word/doubleword with a copy of the
low byte/word/doubleword's sign bit
                ; divides AX, DX:AX, or EDX:EAX over 8, 16, or 32-
bit operand and store in AL, AX, or EAX the result and in AH, DX,
or EDX the reminder
cbw             ; (convert byte to word) extends AL into AH
cwd             ; (convert word to doubleword) extends AX into DX
cdq             ; (convert doubleword to quadword) extends EAX into
EDX
```

```
sbb dest, src       ; subtract both the src opernad and CF from
dest => dest = dest - src - CF
```

Data transfer

```
; No memory to memory
; Destination cannot be EIP, or IP
; No immediate value to segment registers
mov dest, src       ; copies src value into dest  ; src and dest
must be same size
movzx dest, src     ; copies src value into dest with zero
extension ; dest size must be greater than src size ; fills the
rest of the bits with 0's
movsx dest, src     ; copies src value into dest with zero
extension ; dest size must be greater than src size ; fills the
rest of the bits with sign bit
xchg dest, src      ; exchange the values of dest and src ; at
least one operand must be reg ; dest and src must be the same size
; no immediate operands
```

Code Jumping

```
jmp label       ; goes to a label and start executing from

; loops
loop label      ; keeps going to label and dec ecx until ecx
reaches 0 ; does not affect any flags
loope label     ; loop if ecx > 0 and ZF = 1 ; loop if equal ; same
as loopz
loopz label     ; loop if ecx > 0 and ZF = 1 ; loop if zero ; same
as loope
loopne label    ; loop if ecx > 0 and ZF = 0 ; loop if not equal ;
same as loopnz
loopnz label    ; loop if ecx > 0 and ZF = 0 ; loop if not zero ;
same as loopne

; repeat
rep instruction     ; repeates the instruction and decrements ECX
by 1 until ECX = 0
repe instruction    ; repeat while ecx > 0 and ZF = 1
repz instruction    ; same as repe => repeat until ecx = 0 OR ZF =
0
repne instruction   ; repeat while ecx > 0 and ZF = 0
```

```
repnz instruction   ; same as repne => repeat until ecx = 0 OR ZF =
1

call proc       ; goes to proc and start executing from there by
                ; a. Pushes offset of the next instruction on the
stack
                ; b. Copies the address of the procedure into EIP.
ret             ; exit the current proc and return to the prev. EIP
address by pop the top of the stack into EIP
```

Stack Operations

```
push src        ; pushes the value of src at the top of the stack
and decrements ESP by Type src; src can be r/m16, r/m32, or imm32
pop dest        ; pops the value at stack[ESP] into dest and
increments ESP by Type dest ; dest can be r/m16, or r/m32
pushfd          ; pushes the EFLAGS register to the stack, and
decrements ESP by 4
popfd           ; pops 4 bytes from the stack into EFLAGS register
and increments ESP by 4
pusha           ; does the same as pushfd but for the first 16-bits
of EFLAGS
popa            ; does the same as popfd but for the first 16-bits
of EFLAGS

pushad          ; decrements ESP by 32 and pushes the 32-bit
general purpose registers in this order:
                    ; EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
popad           ; pops 32 bytes from the stack into the 32-bit
general purpose registers in this order:
                    ; EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
pusha           ; does the same as pushad but for the 16-bit
general purpose registers
popa            ; does the same as popad but for the 16-bit general
purpose registers
enter bytes, nesting level  ; creates stack frame for a called
procedure
                            ; Pushes EBP on the stack
                            ; Sets EBP to the base of the stack
frame
                            ; Reserves space for local variables
leave           ; terminates the stack frame for a procedure
                ; mov esp,ebp and pop ebp
```

Flags Control

```
stc             ; set carry flag
clc             ; clear carry flag
std             ; set direction flag
cld             ; clear direction flag
```

Boolean Instructions

```
and dest, src       ; performs boolean and between each pair of
bits in dest and src
or dest, src        ; performs boolean or between each pair of bits
in dest and src
xor dest, src       ; performs boolean xor between each pair of
bits in dest and src
not dest            ; performs boolean not on dest ; gets dest 1's
complement ; does not affect any flag
```

Comparison Instructions

```
test dest, src      ; performs AND operation between dest and src
without updating dest value ; only affects the zero flag
                    ; ZF = 1 iff dest AND src = true, else ZF = 0
cmp dest, src       ; performs dest  - src and updates the flags
                    ; if dest == src => for unsigned ZF = 1
; for singed ZF = 1
                    ; if dest < src => for unsigned CF = 1
; for signed SF != OF
                    ; if dest > src => for unsigned ZF = 0, CF = 0
; for signed SF = OF
```

Conditional Jumps

```
; Jcond label
; jump to label if cond is true

; jumps based on flags
jz label            ; jump if zero
jnz label           ; jump if not zero
jc label            ; jump if carry
jnc label           ; jump if not carry
jo label            ; jump if overflow
jno label           ; jump if not overflow
js label            ; jump if sign
jns label           ; jump if not sign
jp label            ; jump if parity
jnp label           ; jump if not parity
```

```
je label            ; jump if equal ; if ZF = 1
jne label           ; jump if not equal ; if ZF = 0
jcxz label          ; jump if CX = 0
jecxz label         ; jump if ECX = 0

; jumps based on unsigned comparisons
ja label            ; jump if above
jae label           ; jump if above or equal
jna label           ; jump if not above
jnae label          ; jump if not above of equal same as jb

jb label            ; jump if below
jbe label           ; jump if below or equal
jnb label           ; jump if not below same as jae
jnbe label          ; jump if not below or equal same as ja

; jumps based on signed comparisons
jg label            ; jump if greater
jng label           ; jump if not greater
jge label           ; jump if greater or equal same as jle
jnge label          ; jump if not greater or equal same as jl

jl label            ; jump if less
jnl label           ; jump if not less same as jge
jle label           ; jump if less or equal
jnle label          ; jump if not less or equal same as jg
```

Shifting

```
; shl dest, count
; dest can be register or memory
; count can only by imm8 or register CL

shl dest, count         ; shift logical left
                            ; preforms logical shift on dest to the
left by count ; fills the lowest bits with 0's ; the last bit
shifted out becomes CF
                            ; shifting left by 1 multiplies the
operand by 2
                            ; shifting left by n multiplies the
operand by pow(2, n)
shr dest, count         ; sheft logical right
                            ; preforms logical shift on dest to the
right by count ; fills the highst bits with 0's ; the last bit
shifted out becomes CF
                            ; shifting right by 1 divides the
operand by 2
```

```
                            ; shifting right by n divides the
operand by pow(2, n)
sal dest, count         ; shift arithmetic left ; performs
arithmetic shift left ; does the same as shl
                            ; shifts dest to the left by count ;
fills the lowest bits with 0's ; the last bit shifted out becomes
CF
                            ; shifting left by 1 multiplies the
operand by 2
                            ; shifting left by n multiplies the
operand by pow(2, n)
sar dest, count         ; shift arithmetic right ; same as shr but
with sign extend instade of zero extending
                            ; preforms arithmetic shift on dest to
the right by count ; fills the highst bits with the sign bit ; the
last bit shifted out becomes CF
                            ; shifting right by 1 divides the
operand by 2
                            ; shifting right by n divides the
operand by pow(2, n)
                            ; divides with preserving the operand
sign
shld dest, src, count   ; Shift Left Double ; shifts dest by count
to the left
                            ; fills the rightmost bits of dest with
the leftmost bits of src
shrd dest, src, count   ; Shift Right Double ; shifts dest by count
to the right
                            ; fills the leftmost bits of dest with
the rightmost bits of src
```

Rotating

```
; rol dest, count
; dest can be register or memory
; count can only by imm8 or register CL

rol dest, count     ; Rotate left ; rotates each to the left by
count
                        ; Highest bit is copied into the Carry Flag
and into the Lowest Bit
ror dest, count     ; Rotate right ; rotates each bit to the right
by count
                        ; Lowest bit is copied into the Carry Flag
and into the Highest Bit
rcl dest, count     ; Rotate Carry Left rotates each bit to the
left by count
```

```
                        ; Copies the Carry flag to the least
significant bit
                        ; Copies the most significant bit to the
Carry flag as if the carry flag is part of the destination operand
rcr dest, count         ; Rotate Carry Right rotates each bit to
the right by count
                        ; Copies the Carry flag to the most
significant bit
                        ; Copies the least significant bit to the
Carry flag as if the carry flag is part of the destination operand
```

String Primitive Instructions

```
; ESI and EDI are automatically incremented or decremented
; The Direction flag controls the incrementing or decrementing of
ESI and EDI
; DF = 0 : increments ESI and EDI => forward
; DF = 1 : decrements ESI and EDI => backward
; DF can by explicitly changed using cld and std
```

- Data Transfer

  ```
  movsb               ; copies a byte         ;
  increments/decrements by 1
  movsw               ; copies a word         ;
  increments/decrements by 2
  movsd               ; copies a double-word  ;
  increments/decrements by 4
  ; copy data from the memory location pointed by ESI to the
  memory location pointed by EDI
  ```

- Comparisons

```
; compare esi, edi
cmpsb               ; compares bytes            ;
increments/decrements by 1
cmpsw               ; copmares words            ;
increments/decrements by 2
cmpsd               ; compares double-words     ;
increments/decrements by 4
; ____________________________________________
; Compare a value in AL/AX/EAX to a byte, word, or doubleword,
respectively, addressed by EDI
scasb               ; compares bytes            ;
increments/decrements EDI by 1
scasw               ; copmares words            ;
increments/decrements EDI by 2
scasd               ; compares double-words     ;
increments/decrements EDI by 4
```

- Storing

```
; copy from registers to memory location
; increments/decrements EDI by 1/2/4
stosb               ; stores AL in EDI offset   ;
increments/decrements EDI by 1
stosw               ; stores AX in EDI offset   ;
increments/decrements EDI by 2
stosd               ; stores EAX in EDI offset  ;
increments/decrements EDI by 4
```

- Loading

```
; copy from memory location at ESI into registers
lodsb               ; load a byte from ESI into AL          ;
increments/decrements ESI by 1
lodsw               ; load a word from ESI into AX          ;
increments/decrements ESI by 2
lodsd               ; load a double-word from ESI to EAX    ;
increments/decrements ESI by 4
```

Irvine I/O Procedures.

- Read

```
call ReadChar       ; reads a single character from the keyboard
and store it in AL reigester
call ReadString     ; reads a string from the keyboard, stops
when the user presses Enter key.
                    ; It takes the offset of a buffer in EDX and
the maximum number of characters the user can enter +1 in ECX
call ReadDec        ; reads 32-bit unsigned decimal integer and
stores it in EAX
call ReadInt        ; reads 32-bit signed dicemal integer and
stores it in EAX
call ReadHex        ; reads 32-bit hex integer and stores it in
EAX
```

- Write

```
call WriteChar      ; writes a single character stored in AL to
the console
call WriteString    ; writes to the console a null-terminated
string whose offset stored in EDX
call WriteDec       ; writes 32-bit unsigned decimal integer
stored in EAX
call WriteInt       ; writes 32-bit signed decimal integer
stored in EAX
call WriteHex       ; writes 32-bit hex integer stored in EAX
call Crlf           ; prints new line to the console window
call DumpMem        ; display a range of memory in Hex starting
from address in ESI, number of items in ECX and item size in EBX
```