CS253 Architectures II

Lecture 4

Assembly Language

Charles Markham

Types of Instruction

Binary Arithmetic	:	Adding and Subtracting
Decimal Arithmetic	:	Adjust binary to BCD
Logical	:	AND, OR, Shifting
Data Transfer	:	Move (mov), Load (ld)
Stack	:	Storing on the stack
Control Transfer	:	Function calls, jumps
String	:	Text arrays
Pointer	:	
Input/Output	:	Reading and writing to ports
Prefix	:	Modifies all other commands
System	:	CPU configuration settings
Floating point	:	The 80837 co-processor commands

Binary Arithmetic

ADD: Integer addition
ADC: Add with Carry
SUB: Subtract with borrow
CMP: Compare integers

Flags affected
AF: Carry out of auxiliary nibble, decimal arithmetic
CF: Carry flag, used for greater than less than
OF: Overflow, result too large or too small
PF: Parity, 1even no. of bits, 0 odd no. of bits
SF: When set it indicates result is negative
ZF: Zero flag, set if result is zero

INC: Increment by 1
DEC: Decrement by 1

Flags affected
As above except CF

DIV: Unsigned divide
IDIV: Signed divide
MUL: Unsigned multiply
IMUL: Signed multiply

Flags affected
Note only one operand, result always in AX

Binary Arithmetic

OPCODE *destination, source*

ADD	AX,10h	Add 16 to AX
SUB	AX,[BX]	Subtract the number pointed to by BX from AX Result is put back in AX.
ADC	AX,10h	Add 16 to AX if CF=0, Add 17 to if CF=1
CMP	AX,10	Subtracts 10 from AX, AX stays the same only the flags are affected. Compare is used to test two numbers without changing either of them. (Very useful command).

Binary Arithmetic

OPCODE *destination, source*

INC BX Increase BX by 1, flags are affected, BX++

DIV 10 AX=AX/10

A reflection of just how CISC the 80386 is one of the few microprocessors to have this function.

Remainder in DX.

MUL BX AX=AX*BX

Decimal Arithmetic

AAA: ASCII Adjust after addition.

AAD: ASCII Adjust before division.

AAM: ASCII Adjust after multiply.

AAS: ASCII Adjust after subtraction.

DAA: Decimal adjust after addition.

DAS: Decimal adjust after subtraction.

Note: These commands are mentioned for the sake of completeness, don't worry greatly about them.

AAA (ASCII Adjust for Addition)

AL=00110101, 35H, ASCII ‘5’, 53d

BL=0011001, 39H, ASCII ‘9’, 57d

ADD AL,BL ;AL=01101110=6EH,110d (Not 14)

AAA ;AL=0000 0100, “0,4” CF=1

5+9 is 14

The CF can be rotated right 4 times and then ORed with AL to produced packed BCD

Packed decimal (Example)

Byte

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1

$\underbrace{1\ 0\ 0\ 0}_{8}\ \underbrace{1\ 0\ 0\ 1}_{9}$

1 Nibble (1/2 byte) per decimal digiti

Some experiments on AAA etc

AAD: Adjust After Addition (single digit)

First digit of result put in AL, if second digit not zero ah is incremented.

```
mov  al,’5’  ;al=35h
add  al,‘7’  ;al=35h+37h=6Ch
aaa          ;al=2, ah=ah+1
```

AAS:Adjust After Subtraction

Similar to above but ah=ah-1

Some experiments on AAA etc

AAD: Adjust Before Division, converts two BCD no.s to a binary number.

Al digit 1, Ah digit 2, each unpacked BCD

AAD: AL=10*AH+AL, AH=0

AAM: Adjust after multiplication

Two unpacked BCD digits are multiplied, the result in ax is in the range [0,81]

AAD: AL=AL mod 10, AH=AL div 10

Single digit multiply demo

```
                          .STARTUP

0017  B0 05   Start:  mov AL,5
0019  B7 09           mov BH,9
001B  F6 E7           MUL BH        ; AX=5*9=45
001D  D4 0A           AAM           ; Convert to unpacked BCD
                                    ; AL=5, AH=4
001F  0D 3030         OR AX,3030H   ; ADD 30 Hex, AL=35H, AH=34H

0022  8A D4           mov dl,ah     ; store ax in bx
0024  8A F0           mov dh,al

0026  B4 02           mov ah,02h    ; ax=02 Print, dl,4
0028  CD 21           int 021h;

002A  8A D6           mov dl,dh     ;Print, dl,5
002C  CD 21           int 021h

                          .EXIT
```

The above program multiplies any single digit decimal 0*0 to 9*9

Note: Start concentrating again!

Logical Instructions

AND:	Boolean AND
OR:	Boolean OR
XOR:	Exclusive OR
BT:	Bit test result in CF
BTC:	Bit test and then complement bit tested
BTR:	Bit test and then reset bit tested
BTS:	Bit test and set the bit tested
BSF:	Bit scan forward until bit set
BSR:	Bit scan reverse until bit set

Logical Instructions

MSB(15) LSB(0)
AX=00010000-00000010

BSF AX,DX

Result : DX=1

MSB(15) LSB(0)
AX=00010000-00000010

BSR AX,DX

Result : DX=3

OR Function often used to convert BCD to ASCII

AL=00001001, 9 Decimal

OR AL,30H

AL=0011 1001, 57 Decimal code for ‘9”

Logical Shift

SHL: Shift Left Logical

SHR: Shift Right Logical

SAL: Shift Arithmetic Left

SAR: Shift Arithmetic Right

ROL: Rotate left

ROR: Rotate right

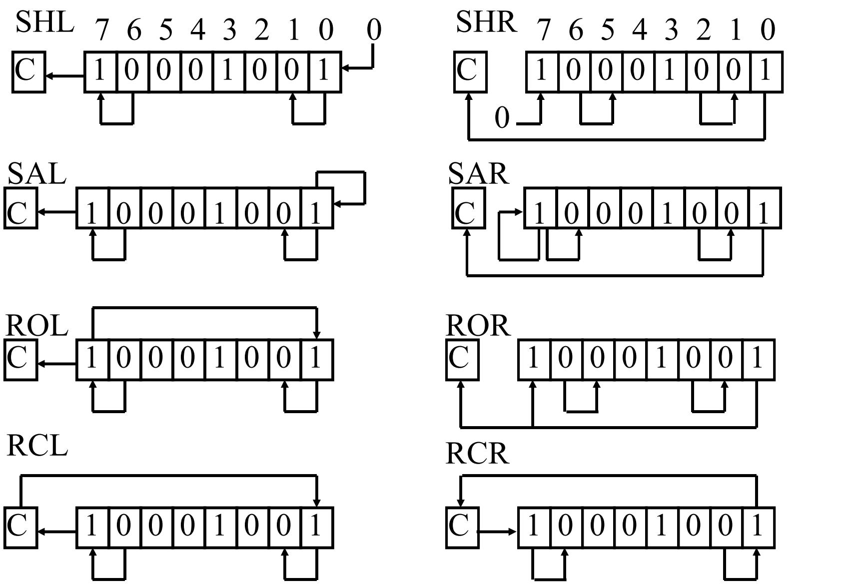
RCL: Rotate through carry left

RCR: Rotate through carry right

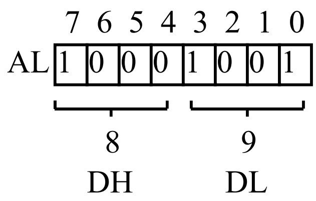
SHLD: Shift left double

SHRD: Shift right double

Logical Shift



AND can be used to mask bits




```
AL=10001001
MOV   DL,AL
AND   DL,00001111b
;DL Equals 9

MOV   DH,AL
AND   DH,11110000b
SHR   DH,4
;DH Equals 8
```

Control Transfer

Control Transfer functions affect the flow of program execution. Normally the IP pointer incremented as each instruction is executed.

Branch Instructions: Change the IP (Instruction pointer)

Call Instructions: Store the current IP on the stack then change the IP.

Note: The instruction pointer could be considered as the line of code currently being executed. It is incremented automatically by the microprocessor after each instruction is executed.

Branch Control Transfer Functions

Jump *offset*

JA	Jump above CF=0,ZF=0	JLE	Jump less or equal SF!=0F, ZF=1
JAE	Jump above or equal CF=0	JNA	Jump not above
JB	Jump below CF=1	JNAE	Jump not above or equal
JBE	Jump below or equal CF=1 ZF=1	JNB	Jump not below
JC	Jump if carry CF=1	JNBE	Jump not below or equal
JCXZ	Jump if CX=0	JCXZ	Jump if CX=0
JZ	Jump if zero ZF=1	JNC	Jump if no carry CF=0
JG	Jump Greater SF=0, ZF=0	JNE	Jump not zero ZF=1
JGE	Jump greater or equal SF=0	JNG	Jump not greater SF!=0, ZF=1
JL	Jump less SF!=0F, ZF=1	JNGE	Jump not greater or equal (JL)

Branch Control Transfer Functions

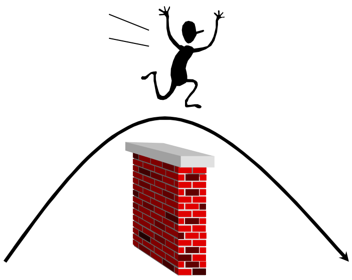
JNL	Jump not less (JGE)
JNLE	Jump not less or equal JG
JNO	Jump no overflow OF=0
JNP	Jump no parity PF=0
JNS	Jump if no sign SF=0
JNE	Jump if not equal ZF=0
JO	Jump if overflow OF=1
JP	Jump if parity
JPE	Jump if parity even PF=1
JPO	Jump if parity odd PF=0
JS	Jump if Sign SF=1
JE	Jump if equal ZF=1

These jumps normally follow a CMP command

```
        mov ax,20
        mov bx,10
        cmp ax,bx
   NC   jnc     over   C
        ;print bx greater than ax
        jmp     pass
over    ;print bx less than ax
pass    ;next bit of code
```

Branch control

Conditional jumps are short jumps, the operand is a single byte that allows a jump back of -128 or forward of +127. Thus you can't jump very far using conditional jumps. Offsets for jumps are counted from the byte after the jump.



Compiled Branch Code

0017	B4 02	start:	mov ah,02h	; ax=02 Print
0019	3C 0A		cmp al,10	If Al<10, print C
001B	72 04		jc lab1	AL=10 or above print NC
001D	B2 4E		mov dl,'N'	
001F	CD 21		int 021h;	+4 Bytes
0021	B2 43	lab1:	mov dl,'C'	
0023	CD 21		int 021h	
0025	EB 04		jmp quit	-9 Bytes 255-9=246, F6h
0027	90		nop	
0028	90		nop	
0029	74 F6		jz lab1	
002B	90	quit:	nop	

The unconditional jump

The unconditional jump can jump much bigger distances.

The compiler will used code with an offset address for small jumps (-128,+127).

A direct address jump is possible allowing a jump of +/-32K

Inter segment jumps are possible.

```
JMP    Label
JMP    Label.seg
```

JMP AX — Jumps using registers are allowed (be careful where you jump!, this needs checking!!!!)

LOOPS

The for(x=0; x<3; x++) of the assembly world

LOOP	Loop if CX not zero
LOOPNZ	Loop while not equal (ZF=0) and CX not zero
LOOPZ	Loop while zero (ZF=1) and CX not zero

CX=CX-1 If CX is not equal to zero and (ZF=0), else

IP=IP+offset

On Z80A this command is djnz (decrease and jump if not zero)

Using loops as delays

```
         MOV CX, N (8)               ; 4   C0=4
WASTE    NOP                   ; 3     ┐
         NOP                   ; 3     ├ CBK=23
         LOOP  WASTE           ; 17 or 5 ┘
```

Clock cycles required to complete the instruction.

$$C_T = C_0 + N(C_{BK}) - C_{CR}$$

C_T is the desired time delay in clock cycles,

C_0 is the overhead

N is the number of times to go around the loop

-12 since 17-5=12 cycles are saved last time through.

Calculating the delay

On a 100MHz machine $T_{Clock}=10^{-8}$S

To create a delay of T_{Delay} =25uS=2.5x10^{-5}S

$$N = \frac{C_T - C_0 + 12}{C_{BK}} \qquad C_T = \frac{T_{Delay}}{T_{Clock}} = \frac{2.5\times10^{-5}}{10^{-8}} = 2500$$

$$N = \frac{2500 - 4 + 12}{23} = 109$$

T_{clock} = Time for one Clock Cycle

mov cx,109   or mov cx,6Ch

Longer delays

```
      mov   bx,count1
lp1:  mov   cx,count2
lp2:  loop  lp2
      dex   bx
      jnz   lp1
```

Use double or triple nested loops to create a long delay.

dec or dex ?

Consider the delay introduced by the inner loop first. Treat this inner loop as a single instruction in your calculation of the delay constant *count1*.

Speed Test

```
        .STARTUP
        mov ah,02         ;Print S
        mov dl,'S'
        int 021h

        mov   bx, 30000   ;4

back2:  mov   cx, 30000   ;4
back1:  nop               ;3
        loop  back1       ;17 or 5

        dec   bx          ;2
        jnz   back2       ;16 or 4

        mov ah,02         ;Print F
        mov dl,'F'
        int 021h
        .EXIT
```

Inner loop

$$C_T = C_0 + N(C_{BK}) - 12$$
$$C_T = 4 + 30000(20) - 12$$
$$C_T = 599992$$

Inner loop

$$C_T = C_0 + N(C_{BK}) - 12$$
$$C_T = 4 + 30000(599992 + 2 + 16) - 12$$
$$C_T = 1.8 \times 10^{10} \text{ Clock cycles}$$

Program took 15 seconds to run

$$Clock\_rate = \frac{1.8 \times 10^{10}\,cycles}{15s}$$

$$Clock\_rate = 1.2 \times 10^{9} = 1200MHz$$

$$Rated\_speed = 300MHz$$

How is computer doing four instructions at a time?

```
         .STARTUP
         mov ah,02          ;Print S
         mov dl,'S'
         int 021h

         mov   bx, 30000    ;4

back2:   mov   cx, 30000    ;4
back1:   nop                ;3
         loop  back1        ;17^ or 5v

         dec   bx           ;2
         jnz   back2        ;16^ or 4v

         mov ah,02          ;Print F
         mov dl,'F'
         int 021h
         .EXIT
```

A method used to print numbers

A=12345

A is in range [0,65535]

C=5

D=10000

B=|A/D|=1, print ASCII character ‘1’

A=A-(D*B), A=2345

D=D/10, D=1000

C=C-1, C=4

Repeat while >0

Note: A is the number to be printed

D is the divisor, 10000, 1000, 100, 10, 1

C is a digit counter

Print AX in decimal

```
        mov     AX,12345 ;
        call    Print
        .EXIT

Print:  push bx     ;Store all registers
        push cx
        push dx

        mov DX,10000
        mov CX,5

NXTCH:  push cx
        push ax
        push dx

        mov  cx,dx    ; al=ax/dx
        mov  dx,0h
        div  cx
        push ax

        mov  bl,al    ; bx=al*dx
        mov  bh,0h
        mov  dx,0h
        mov  ax,cx
        mul  bx
        mov  bx,ax

        pop dx              ;print dl in ASCII
        or  dl,30h
        mov ah,02h
        int 021h

        pop  ax             ;dx->ax
        mov  dx,0h          ;dx/=10
        mov  cx,10
        div  cx
        mov  dx,ax

        pop  ax             ;ax-=bx
        sub  ax,bx
        pop  cx

        loop NXTCH          ;next digit

        pop  dx             ;restore registers
        pop  cx
        pop  bx
        ret
```

Another method to print numbers

A=12345

C=5

A=A/10

Push ASCII 5 on the stack etc

C=C-1

Repeat while C>0

C=5

Pop ASCII char

C=C-1

Repeat while C>0

This approach may be studied in labs.