Announcement

- Last time
  - Introduction to Assembly
- Today
  - Review and wrap up the assembly language for addressing and transferring data

Review of Address Modes

- Register addressing: MOV AX, BX ; <destination> , <source>
- Immediate addressing: MOV AX, 100H

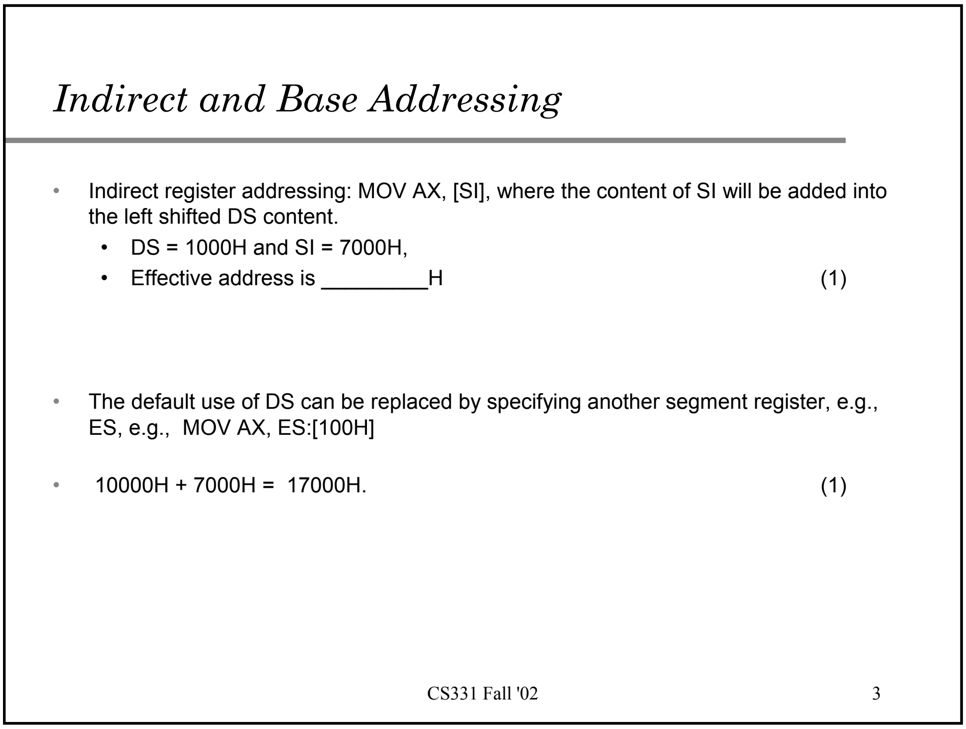
Suppose that we want to move 10H to address 1200H and the segment starts at 1000H. Initially DS = 1000H

- MOV AX, 10H
- MOV [200H], AX

is it correct? If not, what is a correct solution?

- MOV AX, 100H
- MOV DS, AX
- MOV AX, 10H
- MOV [200H], AX

- Intel Assembly looks at the register and decides if it is a byte, 16-bit word, or 32 bit word operation (e.g., AL (or AH): 8bit, AX: 16 bit and EAX: 32 bit).
- ATT assembly uses MOVB, MOVW etc and address the entire register e.g. %eax

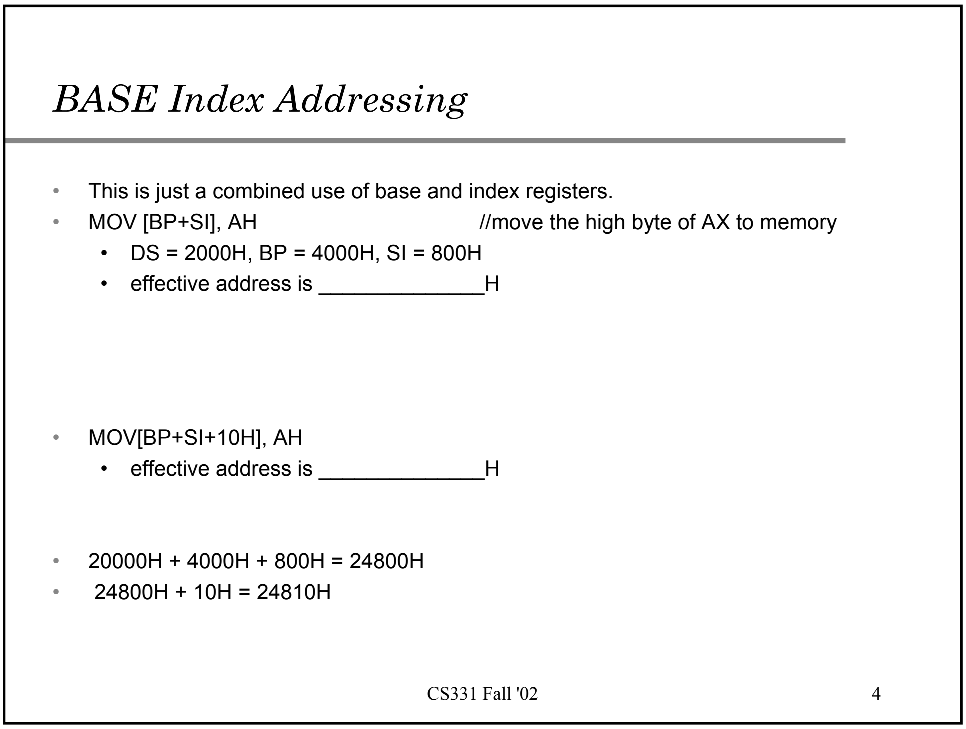


Indirect and Base Addressing

- Indirect register addressing: MOV AX, [SI], where the content of SI will be added into the left shifted DS content.
    - DS = 1000H and SI = 7000H,
    - Effective address is _________H (1)

- The default use of DS can be replaced by specifying another segment register, e.g., ES, e.g., MOV AX, ES:[100H]

- 10000H + 7000H = 17000H. (1)



BASE Index Addressing

- This is just a combined use of base and index registers.
- MOV [BP+SI], AH //move the high byte of AX to memory
    - DS = 2000H, BP = 4000H, SI = 800H
    - effective address is ______________H

- MOV[BP+SI+10H], AH
    - effective address is ______________H

- 20000H + 4000H + 800H = 24800H
- 24800H + 10H = 24810H

Stacks

- The default stack segment register, SS, plays the role of the default data segment register DS
- The stack pointer register, SP, provides the offset.
- PUSH AX, pushes 2 bytes from AX on to the stack and the value of SP = SP - 2
- PUSH EAX pushes 4 bytes from EAX on to the stack and the value of SP = SP - 4

- POP AX loads the data from the stack from the stack to AX and SP = SP + 2
- POP EAX loads the data from the stack from the stack to EAX and SP = SP+ 4

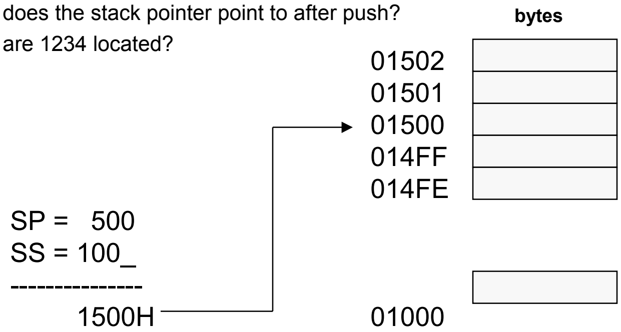
More Pushes and Pops

• PUSHA	Pushes all 16 bit registers
• POPA	POP all 16 bit registers
• PUSHAD	Pushes all 32 bit registers
• POPAD	Pop all 32 bit registers
• PUSHF	Pushes the 16 bit flag register (8086 - 80286)
• POPF	
• PUSHFD	Pushes the 32 bit Extend flag register (386 -)
• POPFD	

Stack

Suppose that

- the stack segment starts at location 1000 H, that is, SS = 100 H
- Currently, SP = 500 H, AX = 1234 H
- PUSH AX
- Where does the stack pointer point to after push?
- Where are 1234 located?



bytes

01502
01501
01500
014FF
014FE

SP =   500
SS = 100_

1500H

01000

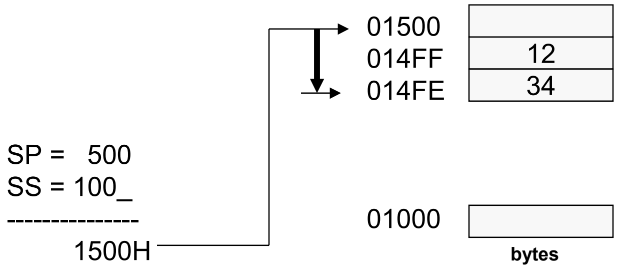
•PUSH AX, pushes 2 bytes from   AX on to the stack and the value of SP = SP - 2

Stack

Suppose that

- the stack segment starts at location 1000 H, SS = 100 H
- the stack size 500H, SP = 500 H
- AX = 1234 H

- What does PUSH AX looks like on the stack?



01500
014FF 12
014FE 34

SP =   500
SS = 100_

1500H

01000

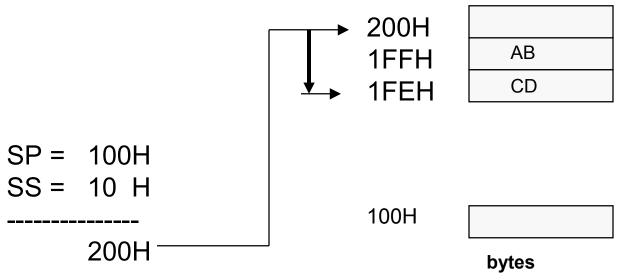
bytes

Stack: Class Exercise

Suppose that we want that

- the stack segment starts at location 100 H
- the stack size 100H
- AX = ABCD H

- What does PUSH AX looks like on the stack?



200H
1FFH
1FEH
AB
CD

SP = 100H
SS = 10 H

200H

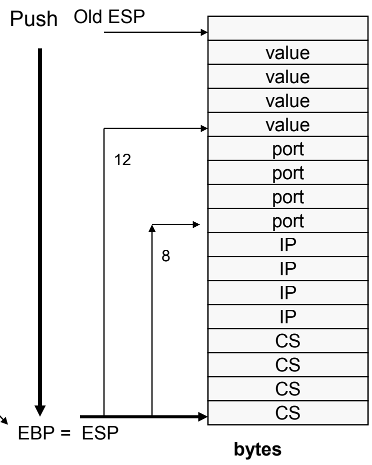
100H
bytes

Inline Function Calls : Intel

```
/* hardware_device_output.c */
hw_out(port, val)
unsigned int port;
char val;
{asm{
//parameters such as "port" and "val" are translated into an offset to the value of EBP
     mov EDX, port[EBP] //copy port address from stack to register EDX
                        //EBP is used as the stack pointer
                        //port address pushed on to stack when function was called

     mov AL, val[EBP] //copy val to accumulator
     out  DX, AL //send val to the location pointed by port address stored in DX.
    }
}
```

The Stack After Call HW_Out under GCC

```
main:
...
0000004e 00000016       MOVSX        EAX,BYTE PTR -5[EBP]
00000052 0000001a       PUSH         EAX   //push val on stack
00000053 0000001b       MOV          EAX,-4[EBP]
00000056 0000001e       PUSH         EAX  //push port on stack
00000057 0000001f       CALL         hw_out
                      //Call    pushes IP (4 byte)
                      //        and    CS (4 bytes)  on Stack
                     // the port is now 8 bytes from ESP

...

     hw_out:
00000068 00000000       PUSH         EBP
00000069 00000001       MOV          EBP,ESP
0000006b 00000003       SUB          ESP,0x0
00000071 00000009       MOV          EDX,8[EBP]
00000074 0000000c       MOV          AL,12[EBP]
00000077 0000000f       OUT          DX,AL
00000078 00000010       LEAVE
00000079 00000011       RET
0000007a 00000012       NOP
0000007b 00000013       NOP
```



- //Note: val is 1 byte and port is 2 byte. But the compiler uses 4 bytesfor each variable anyway.

Inline Function Call: AT&T

We want to perform “b = a”, where “a” and “b” are referred to as input and output variables respectively.

```
{ int a = 10, b = 5;                       /* declare variables in C */

   /* when use inline assembly, add an EXTRA % to register names */
   asm ("movl %1, %%eax;           /* move register %1 to eax */
        movl %%eax, %0;"           /*  move eax value to register %0 */

        /* Specify register(s) used for output variable(s) */
        : "=r"(b)                  /* ask compiler picked ANY register for “b”, referred to as %0  */
                                             /* “r” means let compiler pick a reg. */
                                             /*  “=“ means output reg. */
        /* specify register(s) used for input variable(s) */
        : "r"(a)                   /*  ask compiler pick ANY register for “a”, referred to as %1  */

        : "%eax");                  /* clobbered register, i.e. registers used in addition to those
                                      used for input and out registers  */
        //we do not need to use eax. The use here is purely for illustration of the syntax.
}
```

AT&T Assembly - 2

```
{ int a = 10, b = 5;                        /* declare variables in C */

      asm ("movl %1, %%eax;          /* move register %1 to eax */

            movl %%eax, %0;"           /*  move eax value to register %0 */
             : "=r"(b)                      /* ask compiler picked ANY register for “b”, referred to as %0  */
             : "r"(a)                        /*  ask compiler pick ANY register for “a”, referred to as %1  */
             : "%eax");                     /* clobbered registers*/
```

- Question 1: what is the value of register %0 just after the asm call.

- Question 2: what if we add a line “movl 20H %1”

- Q1: undefined.
- Q2: %1 is specified as input registered and it should NOT be modified. Even if it is modified, the value of this register will not be copied back to the memory location of “a”, because _________

Inline I/O function Calls

```
void hw_out(unsigned int port, unsigned char val)
{

  /* “volatile” tells compiler to do things as is, no optimization tricks please */
  /*  output a byte in Reg %0  to a word specified by Reg %1  */
   __asm__ __volatile__ ("outb %b0, %w1"

                        : /* No output variable */

                        /*    registers used to store input variables */
                        : "a" (val), "d" (port)
                         /*  eax is used to store “val”. This is referred to as Reg %0 */
                         /*  edx is used to store the port number, referred to as Reg %1 */
                         );
```

Summary

- We have reviewed the Intel addressing and data transfer instructions that are commonly used in embedded device I/O.
- In next two classes, we will review interrupt handling

Appendix:

- main()
- {   unsigned int port = 0x300;
- char val = 0x8;
- hw_out(port, val);}

- hw_out(port, val)
- unsigned int port;
- char val;
- {
- asm   {
- mov EDX, port[EBP]
- mov AL, val[EBP]
- out DX, AL   }}

Appendix

```
main:
00000038 00000000    PUSH     EBP
00000039 00000001    MOV      EBP,ESP
0000003b 00000003    SUB      ESP,0x8
0000003e 00000006    CALL     __main
00000043 0000000b    MOV      DWORD PTR -4[EBP],0x300
0000004a 00000012    MOV      BYTE PTR -5[EBP],0x8
0000004e 00000016    MOVSX    EAX,BYTE PTR -5[EBP]
00000052 0000001a    PUSH     EAX
00000053 0000001b    MOV      EAX,-4[EBP]
00000056 0000001e    PUSH     EAX
00000057 0000001f    CALL     hw_out
0000005c 00000024    ADD      ESP,0x8
0000005f 00000027    LEAVE
00000060 00000028    RET
00000061 00000029    LEA      ESI,0[ESI]
00000064 0000002c    ADD      [EAX],AL
00000066 0000002e    ADD      [EAX],AL
      hw_out:
00000068 00000000    PUSH     EBP
00000069 00000001    MOV      EBP,ESP
0000006b 00000003    SUB      ESP,0x0
00000071 00000009    MOV      EDX,8[EBP]
00000074 0000000c    MOV      AL,12[EBP]
00000077 0000000f    OUT      DX,AL
00000078 00000010    LEAVE
00000079 00000011    RET
0000007a 00000012    NOP
0000007b 00000013    NOP
```