The Stack and Introduction to Procedures

Module 8

CS 272

Sam Houston State University

Dr. Tim McGuire

The Stack

- A data structure in which items are added and removed only from one end (the "top")
- A NASM program sets aside a segment for the stack
- **SS** will contain the segment number of the stack segment -- **SP** will be initialized to the stack size (**FFFEh** under in a .com program)
- The stack grows from higher memory addresses to lower ones

Why Stack?

- The 8086 processor has stack instructions
- Procedure calls use the stack for return addresses
- It is convenient to have one around for temporary storage

Where Stack?

- All executables must define a stack segment
  - The stack is an array of bytes accessed via the stack segment register and an offset
- SS points to the beginning of this memory area
- SP is the offset to the top of the stack
  - The loader sets these registers before execution begins

How Stack?

- The stack grows backwards through memory
  - |--unused---xxxxxxUSEDxxxxxxxxx|
  - |-----------^SP marks top of stack
  - |<SS marks start of stack storage
  - |<---Declared Stack Size------>|
- Push decrements stack pointer 
  Pop increments stack pointer 

PUSH and PUSHF

- PUSH *source*
  - source is any 16-bit general or segment register or the address of a word
- PUSHF
  - Pushes the FLAGS register onto the stack
- A PUSH instruction subtracts 2 from SP and then stores the *source* word at SS:SP

POP and POPF

- POP *destination*
  - destination is any 16-bit general or segment register or the address of a word
- POPF
  - Sets the FLAGS register to the value stored at the top of the stack
- A POP instruction copies the word at SS:SP to *destination*, then adds 2 to SP

Stack example

```
push    ax              ;Save ax and bx
push    bx              ;     on the stack

mov     ax, -1          ;Assign test values
mov     bx, -2
mov     cx, 0
mov     dx, 0

push    ax              ;Push ax onto stack
push    bx              ;Push bx onto stack
pop     cx              ;Pop cx from stack
pop     dx              ;Pop dx from stack

pop     bx              ;Restore saved ax and bx
pop     ax              ;     values from stack
```

Stack Over/Underflow

- The processor does not check for either illegal condition
  - Programs may include code to check for stack errors
  - Overflow occurs when SP is smaller than the address of the start of the stack segment
    - Usually this means SP is decremented past 0!
  - Underflow occurs if SP gets bigger than its starting value

Procedure Declaration

- Under NASM, procedures are simply a block of code with a label at the first instruction

```
name:
        ; body of procedure
        ret
```

The CALL Instruction

- CALL invokes a procedure
- CALL has two forms, ***direct***

  `call` ***`name`***

  where ***name*** is the name of a procedure, and

  indirect

  `call` ***`address_expression`*** (not generally recommended)

  where ***`address_expression`*** specifies a register or memory location containing the address of a procedure

Executing a CALL

- The return address to the calling program (the current value of the IP) is saved on the stack
- IP get the offset address of the first instruction of the procedure (this transfers control to the procedure)

The RET instruction

- To return from a procedure, the instruction

  `ret` ***`pop_value`***

  is executed
- The integer argument ***`pop_value`*** is optional
- **`ret`** causes the stack to be popped into IP
- If ***`pop_value`*** *N* is specified, it is added to SP -- in effect removes *N* additional bytes from the stack

Inter-Procedure Communication

- Shared storage
  - The data segment is accessible to all procedures in the current program
- Registers
  - Load registers with arguments (or argument addresses)
  - Store return values in registers
- Place argument information on the stack

Examples of Procedures

- See binbin2.asm and hexbin.asm (these are available on the 272 course page)