Paper 102: Programming & Problem solving through C

Lecture-25:ROM BIOS

Aiusha V.Hujon, Dept. of Computer Science, SAC

ROM BIOS and DOS functions

- These are already present in memory
- They are more efficient in programming the hardware than directly programming the hardware
- Some of the ROM BIOS functions overlap with the DOS functions
- The ROM BIOS works directly with the hardware whereas DOS functions are built from these basic functions
- However they provide the same services

Aiusha V.Hujon, Dept. of Computer Science, SAC

Accessing ROM BIOS functions

- High level functions can be called by using their corresponding name in the programming code
  - E.g., d=sum();
- ROM BIOS functions do not have names
- Calling them is different
- Every ROM BIOS functions has an address
  - These address are stored in the Interrupt Vector Table (IVT)
- Hence we need to obtain the address of a ROM BIOS function from the IVT and pass the control to this address to execute it

Interrupt vector table

- Thus ROM BIOS are invoke using interrupts
- Each interrupt instruction selects a particular address from the IVT and passes control to this address
- This design makes it simple to access any ROM BIOS functions from a program without knowing the specific memory location
- The ROM BIOS functions are divided into different subject categories
  - Each having its own controlling interrupt
  - E.g., all interrupts related with VDU are group under interrupt number 16, printer is interrupt number 23 etc…

IVT

- When a hardware or software interrupt occurs, the microprocessor stops its current processing activity and execute a small memory resident routine called Interrupt Service Routine (ISR)
- Each interrupt has their corresponding ISR
- Once the ISR has been executed, the microprocessor resumes the task before it was interrupted
- Address of these ISRs are stored in the IVT
- Each address is four byte long and is in the form segment:offset
- The first four byte entry in the IVT is the address of the ISR for interrupt number 1 and so on

Steps performed when an interrupt occurs

- Determine the interrupt number
- Multiply this number by four (since each address is four bytes long)
- Indexed the result into the IVT and determine the address of the corresponding ISR
- Once found, the contents of registers of microprocessors are saved onto a stack
- The registers are set up to run the ISR
- Execute the ISR
- When done, copy the previous values of registers on stack into the register again, so previous work can be resumed

The int86() function

- This functions make a software interrupt occur
  - Hence invoking the ROM BIOS function
- The function needs two arguments
  - Interrupt number corresponding to the ROM BIOS function to be invoked
  - Two union variables
    - First union variable represent value sent to the ROM BIOS routine
    - Second represents values being returned from the ROM BIOS routines
  - The values are passed and returned from ROM BIOS routines through CPU registers
- int86(16,&inregs,&outregs);

CPU registers

- There are 14 registers each with a special purpose
- They are group as under:
- Scratch-pad registers:
  - Also known as data registers
  - There are four in number
  - Use for by programs to temporarily store intermediate results and operands of arithmetic and logical comparison operations
  - AX-accumulator
  - BX-base
  - CX-count
  - DX-data

CPU registers

- There are 14 registers each with a special purpose
- They are group as under:
- Scratch-pad registers
- Segment registers
- Offset registers
- Flags register

Scratch-pad registers

- Also known as data registers
- There are four in number
- Use for by programs to temporarily store intermediate results and operands of arithmetic and logical comparison operations
  - AX-accumulator
  - BX-base
  - CX-count
  - DX-data
- These 16 bits registers are divided into high order 8 bits registers known as AH, BH, CH, DH and low order 8 bits registers known as AL, BL,CL,DL

Segment registers

- The 8086 family refers to any memory location in terms of a 16 bit segment value and an offset
- Specific segments of memory are identified by segment registers and offsets within the segment are identified by the offset registers
  - CS register- code segment
  - DS and ES registers identify the data segment and extra segment
  - SS register identifies the stack segment

Offset registers

- Five offset registers
- These are used in conjunction with the segment registers
- IP-instruction pointer (program counter)
- SP- stack pointer
- BP-base pointer
- SI-source index
- DI-destination index
  - SI and DI are general purpose addressing of data

Flags register

- Used to indicate if a process or operation has executed successfully or not
- There are nine 1 bit flags in the 16 bit flags registers, 7 are unused

WORDREGS

- WORDREGS is a structure consisting of all the registers in their two byte interpretation
- they contain variables ax,bx,cx,dx,si,di,cflag,flags
- Struct WORDREGS
- {   unsigned int ax,bx,cx,dx,si,di,cflag,flags;
- };

BYTEREGS

- BYTEREGS is a structure consisting of AX,BX,CX,DX registers interpreted as 8 one byte variables
- they contain variables al,ah,bl,bh,cl,ch,dl,dh
- Struct BYTEREGS
- {   unsigned int al,ah,bl,bh,cl,ch,dl,dh;
- };

REGS

- The two structures are combined to form a union called REGS
- Hence this union variable can be treated as two byte registers or one byte registers

```
union REGS
{       struct WORDREGS x;
        struct BYTEREGS h;
};
```

These are all stored in “dos.h”

Finding memory size

```
#include<dos.h>
Main()
{
        union REGS inregs,outregs;
        int memsize;
Int86(18,&inregs,&outregs);
Memsize=outregs.a.ax;
Printf("\n memory size is=%d",memsize);
}
```