ECE291

Lecture 7

Go set the table

Lecture outline

- Arrays
- Lookup tables
- Hash tables
- Jump tables
- Where this is leading…

Arrays

- Collection of sequentially addressable equally sized data in memory
- Only first element is labeled
- Index from there given size of each array element

Arrays - declaring

```
;declare and initialize a five element byte array
MyByteArray DB    00h,0afh,12,83,83h

;declare a 50 element word array
MyWordArray1      RESW  50
MyWordArray2      RESB  100
MyWordArray3      RESD  25

;declare and initialize a 16 element char array
MyCharArray DB    “ECE291 Rocks!!!”, 0
```

Arrays - accessing

```
/* C array example */

int intWordArray[25];

for (int i = 0; i < 25; i++) {
  intWordArray[i] = i * i;
/*    Same as:
      *(intWordArray + 2*i) = i * i;       */
}

/* This code segment declares a 25-element word
  array (50 bytes) and fills each element with the
  square of its index */
```

Arrays - accessing

```
;Assembly array example

intWordArray       RESW  25

mov cx, 25
begin_loop:
   mov bx, cx
   imul ax, cx, cx
   dec bx
   shl bx
   mov [intWordArray + bx], ax
loop begin_loop
```

Arrays in 2-dimensions

0	1	2	3	4	5	6	7	8	9	A	B
C	D	E	F	10	11	12	13	14	15	16	17
18	19	1A	1B	1C	1D	1E	1F	20	21	22	23
24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B
3C	3D	3E	3F	40	41	42	43	44	45	46	47
48	49	4A	4B	4C	4D	4E	4F	50	51	52	53

- This is a 12 col by 7 row 2-D byte array
- It has 12 * 7 = 84 = 54h elements
- Addresses increase as you traverse a row
- Each row begins NUMCOLS addresses after the previous row
- Address of element (row, col) = row*NUMCOLS + col

```
NUMCOLS EQU     12
NUMROWS EQU     7
My2DArray RESB NUMCOLS * NUMROWS
```

```
mov bx, 4
mul bx, NUMCOLS
add bx, 6

mov al, [My2DArray + bx]
  ;[My2DArray + 36] → AL
```

Lookup tables

- Lookup tables are special purpose arrays commonly used to convert one data form to another
- A lookup table is formed in the memory as a list of data that is referenced by a procedure to perform conversions

Lookup tables

- Consider movement in a 2-dimensional array by one element to an adjacent element (up, down, left, or right)
  - Array = Matrix of ROWSIZE x COLSIZE elements
  - Pos = Position in array = Row * COLSIZE + Column
  - Dir = Direction (UP=0, RIGHT=1, DOWN=2,LEFT=3)
- Slow and tedious original code compares each possible direction value
- 16 instructions

```
Mbegin
   CMP   [DIR], UP
   JE    MoveUP
   CMP   [DIR], RIGHT
   JE    MoveRT
   CMP   [DIR], DN
   JE    MoveDOWN
   CMP   [DIR], LEFT
   JE    MoveDown

MoveUP
   SUB   [Pos], COLSIZE
   JMP   MDone
MoveRT
   ADD   [Pos],1
   JMP   MDone
MoveDN
   ADD   [Pos],COLSIZE
   JMP   MDone
MoveLT
   SUB   [Pos],1
   JMP   MDone
MDone
```

Lookup tables

- Fast and Compact Table-lookup Code
- Eliminate conditional jumps by defining a table of movements indexed by the Direction
- 4 instructions + 4 word-sized variables

```
Movement
  dw -COLSIZE           ; Movement[0] == Go UP
  dw +1                 ; Movement[1] == Go RIGHT
  dw +COLSIZE           ; Movement[2] == Go DOWN
  dw -1                 ; Movement[3] == Go LEFT
Mbegin
  MOV BX, [DIR]         ; Direction = {0..3}
  SHL BX, 1             ; Index words, not bytes
  MOV AX, [Movement + BX]
  ADD [Pos], AX         ; Position += Movement
MDone:  Finished!
```

Lookup tables

```
DAYS
    PUSH DX
    PUSH SI

    MOV SI, DTAB    ;si points to table
    XOR AH, AH      ;clear AH
    ADD AX, AX      ;double AX
    ADD SI, AX      ;index to right day
    MOV DX, [SI]    ;get string address


    MOV AH, 09h     ;display char magic
    INT 21h

    POP SI
    POP DX
RET
```

```
SUN DB   'Sunday$'
MON DB   'Monday$'
TUE DB   'Tuesday$'
WED DB   'Wednesday$'
THU DB   'Thursday$'
FRI DB   'Friday$'
SAT DB   'Saturday$'

;Lookup table contains addresses of
;day of week strings.  It converts
;0 to "Sunday", 1 to "Monday", etc.

DTAB DW  SUN, MON, TUE,
         WED, THU, FRI, SAT
```

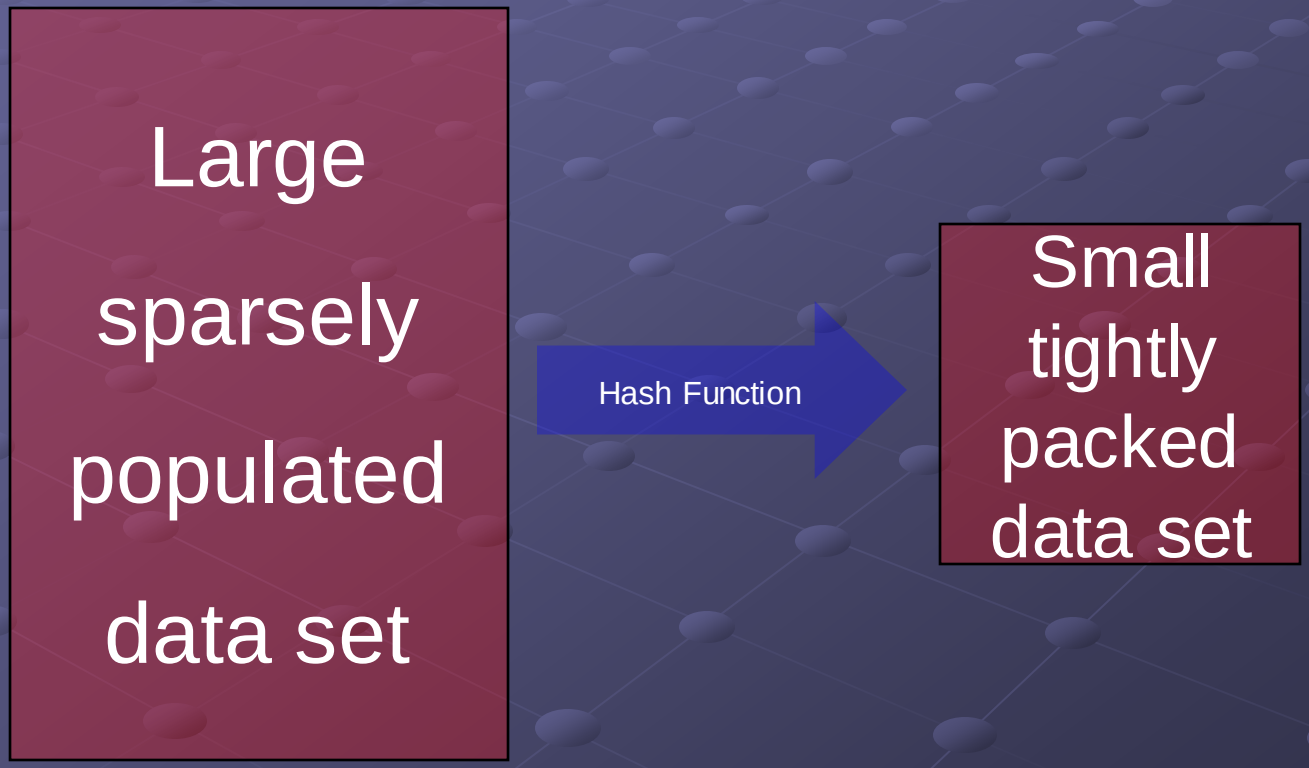
Converts numbers 0 to 6 to days of the week
The number is passed in AL register

Hash functions

- Consider using a lookup table when the input can span a large range of values but is sparsely populated.
- Example:
  - Count incoming packets from each of 100 hosts scattered throughout the Internet using the 32-bit IP source address
  - A Table-Lookup would require 4 billion table entries, most of which would be empty
- This is what we call “A Bad Thing.”

Hash functions

- Hash functions re-map large sparsely populated data sets (like the set of all IP addresses) into smaller more manageable data sets.



Hash functions

- A Hash Function, H(x), can be used to re-map the four-byte (W.X.Y.Z) IP address to asmaller (8-bit) value.
- H(W.X.Y.Z)=W xor X xor Y xor Z

H(0.0.0.0) = 0

H(128.174.5.58) = 17

H(224.2.4.88) = 190

H(141.142.44.33) = 14

Hash Table

Index	IP address
0	0.0.0.0
…	
14	141.142.44.33
…	
17	128.174.5.58
…	
190	224.2.4.88
…	

Hash functions

- Collisions can occur when two inputs map to the same output
  - H(128.174.112.1) and H(128.174.1.112) for example
- Collision resolution—Linear Probing
- Use next-available location in the hash table
- Add an extra column in the table for identifiers or tags to distinguish different entries in the table

```
tag = IP_addressTag
index = h(IP_address);
if (ht[index].tag == IP_addressTag) then
  <Done>
else
  <search the hash table until the entry with
  matching Tag is found>
end if
```

Jump tables

- Suppose table entries are pointers to functions
- BX holds a command in the range of 0..N
- Let BX index a function

```
JFUNCTION                ; Input BX = Command
    SHL BX,1             ; Words, not bytes
    JMP [Jtable+BX]      ; Jump to function

    Jtable
                dw      Funct0
                dw      Funct1
                ..
                dw      FunctN

    Funct0      ;do something ..
                RET
    Funct1      ;do something else ..
                RET
    FunctN      ;do something else ..
                RET
```

Where this is leading…

- Interrupts! These are triggers that cause the CPU to perform various tasks on demand
- Three kinds:
  - Software interrupts
  - Hardware interrupts
  - Exceptions
- Regardless of source, they are handled the same
  - Each interrupt has a unique interrupt number from 0 to 255. These are called interrupt vectors.
  - For each interrupt vector, there is an entry in the interrupt vector table.
  - The interrupt vector table is simply a jump table containing segment:offset addresses of procedures to handle each interrupt
  - These procedures are called *interrupt handlers* or *interrupt service routines*

Where this is leading…

- The first 1024 bytes of memory (addresses 00000 – 003FF) always contain the interrupt vector table. Always.
- Each of the 256 vectors requires four bytes—two for segment, two for offset

Memory address (hex)	Interrupt function pointer
003FC	INT 255
4 * x	INT x
00008	INT 2
00004	INT 1
00000	INT 0

Software interrupts

- The operating system has handlers for many interrupts. These routines provide various built-in functions:
  - Displaying text to the screen (formerly known as “magic”)
  - Opening files
  - Rebooting the system (black magic)
  - Changing video modes
- Accessed with the INT instruction

DOS function dispatcher

- INT 21h is the DOS function dispatcher. It gives you access to dozens of functions built into the operating system.
- To execute one of the many DOS functions, you can specify a sub-function by loading a value into AH just before calling INT 21
- INT 21h sub-functions
  - AH=3Dh: Open File
  - AH=3Fh: Read File
  - AH=3Eh: Close File
  - AH=13h: Delete File (!)
  - AH=2Ah: Get system date
  - AH=2Ch: Get system time
  - AH=2Ch: Read DOS Version
  - AH=47h: Get Current Directory
  - AH=48h: Allocate Memory block (specified in paragraphs==16 bytes)
  - AH=49h: Free Memory block
  - AH=4Ch: Terminate program (and free resources)

System BIOS functions

- All PCs come with a BIOS ROM (or EPROM).
- The BIOS contains procedures that provide basic functions such as bootstraping and primitive I/O.
  - INT 19h: Reboot system
  - INT 11h: Get equipment configuration
  - INT 16h: Keyboard I/O

Video BIOS functions

- Video cards come with procedures stored in a ROM
- Collectively known as the video BIOS
- Located at C0000-C7FFF and holds routines for handling basic video adapter functions
- To execute a function in video BIOS ROM, do an INT 10h with video sub-function number stored in AX
- INT 10h, Sub-function examples
  - AH=0, AL=2h: 80 column x 25 row text display mode
  - AH=0, AL=13h: 320x200 pixel, 256-color graphics display mode

Where is all this stuff???