CS 271 Lecture #4

Introduction to MASM assembly language

MASM instruction types

- Move data
- Arithmetic
- Compare two values
- Conditional/unconditional branch
- Call procedure, return
- Loop control
- I/O (input/output)

MASM Directives

- Tell the assembler how to interpret the code
  - Mark beginning of program segments … e.g.

```
.data
.code
```

  - Mark special labels … e.g.

```
main        proc
varName     DWORD
```

  - etc.

MASM Program Template (link)

```
TITLE Program Template      (template.asm)

; Author:
; Course/project ID                          Date:
; Description:

INCLUDE  Irvine32.inc

    <insert constant definitions here>

.data
    <insert variable definitions here>

.code
main PROC

    <insert executable instructions here>

    exit            ; exit to operating system
main ENDP

    <insert additional procedures here>

END main
```

MASM programming

- **`TITLE`** directive
  - you can put anything you want
  - … but the grader wants to see a meaningful title and the name of the source code file
- `;` identification block
  - technically optional (as are all comments)
  - … but the grader wants to see information
- **`INCLUDE`** directive
  - copies a file of definitions and procedures into the source code
  - use **`Irvine32.inc`** for now

MASM programming

- Global constants may be defined

- `.data` directive
  - marks beginning of data segment
  - variable declarations go here
- `.code` directive
  - marks end of data segment and beginning of code segment
  - `main` procedure defined here (required)
    - other procedures defined here (optional)
    - `main` must have an `exit` instruction
    - all procedures require `PROC` and `ENDP` directives
- `END` directive
  - tells operating system where to begin execution

MASM syntax and style

- MASM is **not** case-sensitive !!
  - constants usually ALL CAPS
- Segments start with `.`
  - `main` should be the first procedure in the `.code` segment
  - beginning of next segment (or `END main`) is end of segment
- Comments start with `;`
  - can start anywhere in a line
  - remainder of line is ignored by the assembler
  - end of line is end of comment
- Use indentation and sufficient white space to make sections easy to find and identify

MASM identifier syntax

- Identifiers: Names for variables, constants, procedures, and labels
- 1 to 247 characters (no spaces)
  - Use concise, meaningful names
- NOT case sensitive!
- Start with letter, _ , @, or $
  - For now, start with letter only
- Remaining characters are letters, digits, or _
- Cannot be a reserved word
  - e.g.: proc, main, eax, … etc.

Memory Locations

- May be named
  - Name can refer to a variable name or a program label
- Interpretation of contents **depends on program instructions**
  - Numeric data
    - Integer, floating point
  - Non-numeric data
    - Character, string
  - Instruction
  - Address
  - etc.

MASM data types syntax

Type	Used for:
BYTE	Character, string, 1-byte integer
WORD	2-byte integer, address
DWORD	4-byte unsigned integer, address
FWORD	6-byte integer
QWORD	8-byte integer
TBYTE	10-byte integer
REAL4	4-byte floating-point
REAL8	8-byte floating-point
REAL10	10-byte floating-point

MASM Data definition syntax

- in the **`.data`** segment
- General form is

```
label       data_type       initializer       ;comment
```

  - **`label`** is the "variable name"
  - **`data_type`** is one of *(see previous slide)*
  - at least one **`initializer`** is required
    - may be **`?`** (value to be assigned later)
- Examples:

```
size      DWORD 100              ;class size
celsius   WORD  -10              ;current Celsius temp
response  BYTE  'Y'              ;positive answer
myName    BYTE  "Wile E. Coyote",0
gpa       REAL4 ?                ;my GPA
```

Data in Memory

- "variables" are laid out in memory in the order declared
- Example:

```
.data
size        DWORD  100              ;class size
celsius     WORD   -10              ;current Celsius
response    BYTE   'Y'              ;positive answer
myName      BYTE   "Wile E. Coyote",0
gpa         REAL4  ?                ;my GPA
```

  - Suppose that the data segment starts at memory address 1000

size	is address 1000	(DWORD uses 4 bytes)
celsius	is address 1004	(WORD uses 2 bytes)
Response	is address 1006	(BYTE uses 1 byte)
myName	is address 1007	(Each character uses 1 byte)
	(Blank spaces and the terminating 0 are characters too!)	
gpa	is address 1022	

Data in Memory

`size`	is address 1000	(DWORD uses 4 bytes)
`celsius`	is address 1004	(WORD uses 2 bytes)
`Response`	is address 1006	(BYTE uses 1 byte)
`myName`	is address 1007	(Each character uses 1 byte)
		(Blank spaces and the terminating 0 are characters too!)
`gpa`	is address 1022	

NOTE:

- Each **name** is a **constant**
  - i.e. the system substitutes the memory address for each occurrence of a name
- The **contents** of a memory location may be **variable**.

Literals

- Actual values, named constants
  - Integer
  - Floating point
  - Character
  - String (only in `.data` segment or named constant)

- Used for:
  - Initializing variables (in the `.data` segment)
  - Defining constants
  - Assigning contents of registers
  - Assigning contents of memory (in the `.code` segment)

MASM Literals syntax

- Integer
  - Optional radix: b, q/o, d, h
    - Digits must be consistent with radix (e.g., 1011b, 235q, 2012d, 30h)
    - Hex values that start with a letter must have a leading 0 (e.g., 0A3h)
      - or use the 0x prefix instead of the radix (e.g., 0xA3)
  - Default is decimal
- Floating-point (decimal real)
  - Optional sign
  - Standard notation (e.g., -3.5 +5. 7.2345)
  - Exponent notation (e.g., -3.5E2 6.15E-3)
  - Must have a decimal point

MASM Literals syntax

- Character
  - Single character in quotes
    - `'a'  "*"  '3'`
    - Single quotes recommended
- String
  - 2 or more characters in quotes
    - `"always",0`
    - `'123 * 654',0`
    - Double quotes recommended
    - Embedded quotes must be different
      - `"It's",0  'Title: "MASM"',0`
  - Strings must be null-terminated
    - Always end with zero-byte

MASM Instruction syntax

- Each instruction line has 4 fields:
  - Label
  - Opcode
  - Operands
  - Comment
- Depending on the opcode, one or more operands may be required
  - Otherwise, any field may be empty
  - If empty opcode field, operand field must be empty

MASM Instruction syntax

- Opcode (specifies what to do)
  - Mnemonic (e.g., ADD, MOV, CALL, etc.)
- Zero, one, or two Operands (specify the opcode's target)
  - different number of operands for different opcodes

```
opcode
opcode    destination
opcode    destination, source
```

MASM Addressing modes

Specific "addressing modes" are permitted for the operands associated with each opcode.

- Basic (used in first programming assignment)
  - Immediate — Constant, literal, absolute address
  - Register — Contents of register
  - Direct — Contents of referenced memory address
  - Offset — Memory address; may be calculated
- Advanced (these will be used later in the course)
  - Register indirect — Access memory through address in a register
  - Indexed — "array" element, using offset in register
  - Base-indexed — Start address in one register; offset in another, add and access memory
  - Stack — Memory area specified and maintained as a stack; stack pointer in ESP register

See the MASM list of instructions