ECE 3120
Computer Systems
Assembly Programming

Manjeera Jeedigunta

http://blogs.cae.tntech.edu/msjeedigun21

Email: msjeedigun21@tntech.edu

Tel: 931-372-6181, Prescott Hall 120

Prev:

- Basic computer concepts
- 68HCS12 addressing modes,instructions

Today:

- Programming Structure
- Assembler Directives

3 Sections of a HCS12 Assembly Program

- Assembler directives
  - Defines data and symbol
  - Reserves and initializes memory locations
  - Specifies output format
  - Specifies the end of a program
- Assembly language instructions
  - HCS12/MC9S12 instructions
- Comments
  - Explains the function of a single or a group of instructions

Fields of a HCS12 Instruction

Instruction

label: opcode operands ;comments

- Label field
  - Optional
  - Starts with a letter and followed by letters, digits, or special symbols (_ or .)
  - Can start from any column if ended with ":"
  - Must start from column 1 if not ended with ":"
- Operation field
  - Contains the mnemonic of a machine instruction or an assembler directive
  - Separated from the label by at least one space
- Operand field
  - Follows the operation field and is separated from the operation field by at least one space
  - Contains operands for instructions or arguments for assembler directives
- Comment field
  - Any line starts with an * or ; is a comment
  - Separated from the operand and operation field for at least one space
  - Optional

Identify the Four Fields of an Instruction

loop ADDA #$40 ; add 40 to accumulator A

(1) “loop” is a label
(2) “ADDA” is an instruction mnemonic
(3) “#$40” is the operand
(4) “add #$40 to accumulator A” is a comment

movb 0,X,0,Y ; memory to memory copy

(1) no label field
(2) “movb” is an instruction mnemonic
(3) “0,X,0,Y” is the operand field
(4) “; memory to memory copy” is a comment

Assembler Directives

- **END**
  - Ends a program to be processed by an assembler
  - Any statement following the END directive is ignored.
- **ORG**
  - The assembler uses a location counter to keep track of the memory location where the next machine code byte should be placed.
  - This directive sets a new value for the location counter of the assembler.
  - The sequence

    ORG $1000

    LDAB #$FF

    places the opcode byte for the instruction LDAB #$FF at location $1000.

dc.b (define constant byte)
db (define byte)
fcb (form constant byte)

- These three directives define the value of a byte or bytes that will be placed at a given location.
- These directives are often preceded by the **org** directive.
- For example,

```
        org  $800
array   dc.b  $11,$22,$33,$44
```

dc.w (define constant word)
dw (define word)
fdb (form double bytes)

- Define the value of a word or words that will be placed at a given location.
- The value can be specified by an expression.
- For example,

```
vec_tab  dc.w      $1234, abc-20
```

fcc (form constant character)

- Used to define a string of characters (a message)
- The first character (and the last character) is used as the delimiter.
- The last character must be the same as the first character.
- The delimiter must not appear in the string.
- The space character cannot be used as the delimiter.
- Each character is represented by its ASCII code.
- Example

```
msg    fcc   “Please enter 1, 2 or 3:”
```

fill (fill memory)

- This directive allows the user to fill a certain number of memory locations with a given value.
- The syntax is **fill value,count**
- Example

  space_line **fill** $20,40

ds (define storage)

rmb (reserve memory byte)

ds.b (define storage bytes)

- Each of these directives reserves a number of bytes given as the arguments to the directive.
- Example

  buffer ds 100

  reserves 100 bytes

Storage

ds.w (define storage word)

rmw (reserve memory word)

- Each of these directives increments the location counter by the value indicated in the number-of-words argument multiplied by two.
- Example

  ```
  dbuf     ds.w  20
  ```

  reserves 40 bytes starting from the current location counter

equu (equate)

- This directive assigns a value to a label.
- Using this directive makes one's program more readable.
- Examples

  ```
  arr_cnt   equ    100
  oc_cnt    equ    50
  ```

loc

This directive increments and produces an internal counter used in conjunction with the backward tick mark (`).

-No need to think up new labels:

```
        loc                                 loc
        ldaa  #2          same as           ldaa  #2
loop`   deca                        loop001 deca
        bne   loop`                         bne  loop001
        loc                                 loc
loop`   brclr  0,x,$55,loop`                loop002  brclr  0,x,$55,loop002
```

Macro

A name assigned to a group of instructions

- Use **macro** and **endm** to define a macro

- Example of macro

```
sumOf3          macro     arg1,arg2,arg3
        ldaa    arg1
        adda    arg2
        adda    arg3
        endm
```

- Invoke a defined macro: write down the name and the arguments of the macro

```
        sumOf3  $1000,$1001,$1002
```

is replaced by

```
        ldaa    $1000
        adda    $1001
        adda    $1002
```

Next…

- Software Development Issues
- Programming Arithmetic