ECE 3120
Computer Systems
HCS12 Assembly Programming
- Review

Manjeera Jeedigunta
http://blogs.cae.tntech.edu/msjeedigun21
Email: msjeedigun21@tntech.edu
Tel: 931-372-6181, Prescott Hall 120
Program Structure

Assembler Directives
  Initializations
  Reserving Memory
  Declaring array elements

Main Logic of the program
Arithmetic Programming

- Addition
- Subtraction
- Multiprecision Addition (Carry Flag)
- Multiprecision Subtraction (Borrow Flag)
- BCD
- Multiplication
- Division
Example 1: Addition of 16 bit numbers

Write a program to add two 16 bit numbers, one of which is $1234 and the other is stored at memory location $1000~1001. Store the result in $2000

```
org $1500
ldd #$1234 ;place 16 bit number in D
            ;D=$1234
addd $1000 ;D + m[$1000]:m[$1001] → D
std $2000 ;Sum → $2000
```
Write a program to add two 16 bit numbers, one of which is at $1000\sim1001$ and the other is stored at memory location $1002\sim1003$ and subtract a 16 bit number that is stored in $1100\sim1101$ from the sum. Store the result in $2000$

```
org $1500
ldd $1000 ;D←m[$1000]:m[1001]
addd $1002 ;D←D+m[$1002]:m[$1003]
subd $1100 ;D←D-m[$1100]:m[$1101]
std $2000 ;m[$2000]←D
```
Example 3: Multiprecision Addition

Write a program to add two 4 byte numbers that are stored at $2000$-$2003$ & $2100$-$2103$ and store the result at $2200$-$2203$.

```
+       +       +
A       A       A       B
$2103$  $2102$  $2101$  $2100$

Sum $\rightarrow$ A  Sum $\rightarrow$ A  Sum $\rightarrow$ D

$2000$
$2001$
$2002$
$2003$

$2200$
$2201$
$2202$
$2203$
```
Example 3: Multiprecision Addition

Write a program to add two 4 byte numbers that are stored at $2000$-$2003$ & $2100$-$2103$ and store the result at $2200$-$2203$

```
org $1000
ldd $2000 ;D ← m[$2000]:m[$2001]
addd $2100 ;D ← D + m[$2100]:m[$2101]
std $2200 ; m[$2200] ← D
ldaa $2002 ; A ← m[$2002]
adca $2102 ; A ← A + m[$2102]
staa $2202 ; m[$2202] ← A
ldaa $2003 ; m[$2003]
adca $2103 ; A ← A + m[$2103]
staa $2203 ; m[$2203] ← A
```
Example 4: Multiprecision Subtraction

Write a program to subtract two 4 byte numbers that are stored at $2000$~$2003$ & $2100$~$2103$ and store the result at $2200$~$2203$

```
org $1000
ldd $2000 ;D←m[$2000]:m[$2001]
subd $2100 ;D←D-m[$2100]:m[$2101]
std $2200 ;m[$2200]←D
ldaa $2002 ;A←m[$2002]
sbca $2102 ;A←A-m[$2102]
staa $2202 ;m[$2202]←A
ldaa $2003 ;m[$2003]
sbca $2103 ;A←A-m[$2103]
staa $2203 ;m[$2203]←A
```
Example 5: Multiplication

Write a program to multiply two **signed 16 bit numbers** that are stored at $2000$-$2001$ & $2100$-$2101$ and store the result at $2200$-$2201$

```
org $1000
ldd $2000 ;D ← m[$2000]:m[$2001]
ldy $2100 ;Y ← [2100]:m[$2101]
emuls
std $2201 ;m[$2201] ← D
sty $2200 ;m[$2200] ← Y
```
Example 6: Division

Write an instruction sequence to divide the signed 16-bit number stored at $1005-$1006 by the signed 16-bit number stored at $1020-$1021 and store the quotient and remainder at $1030-$1031 and $1032-$1033, respectively.

ldd $1005 ; load the dividend into D
ldx $1020 ; load the divisor into X
idivs ; perform signed division
stx $1030 ; storing the quotient
std $902 ; storing the remainder
Example 2.11 Write an instruction sequence to divide the signed 16-bit number stored at $1005$-$1006$ by the signed 16-bit number stored at $1020$-$1021$ and store the quotient and remainder at $1030$-$1031$ and $1032$-$1033$, respectively.

\[
\begin{array}{c}
\text{D} \\
\text{LSB of Dividend} \quad 1006 \\
\text{MSB of Dividend} \quad 1005 \\
\end{array} \\
\begin{array}{c}
\text{X} \\
\text{LSB of Divisor} \quad 1021 \\
\text{MSB of Divisor} \quad 1020 \\
\end{array} \\
= \\
\begin{array}{c}
\text{D} \\
\text{LSB of Rem} \quad 1033 \\
\text{MSB of Rem} \quad 1032 \\
\end{array} \\
\begin{array}{c}
\text{X} \\
\text{LSB of Quotient} \quad 1031 \\
\text{MSB of Quotient} \quad 1030 \\
\end{array}
\]

ldd $1005$ ; load the dividend into D
ldx $1020$ ; load the divisor into X
idivs ; perform signed division
stx $1030$ ; storing the quotient
std $902$ ; storing the remainder
Example 7 : BCD

Write a program to add the 4 digit BCD numbers that are stored at $2000$-$2001$ & $2100$-$2101$ and store the result at $2200$-$2201$

Similar to normal addition, except we have to take care of the decimal adjustment part by using the ‘dAA’ instruction. Works with only reg A so have to work with one byte at a time starting from the LSB

```
org $1000
lda $2001 ;A←m[$2001]
adaa $2101 ;A←A+m [$2101]
da  ;decimal adjust lower byte
staa $2201 ;m[$2201]←A
lda $2000 ;A←m[$2000]
adAA $2100 ;A←A+m [$2100]
da  ;decimal adjust Higher byte
staa $2200 ;m[$2200]←A
```
Example 8: Loops

Write a program to compute the sum of 10, 16-bit unsigned numbers stored at the memory address $1000$-$1020$ and store the result in $1100$-$1103$

N equ 10 ; array count
org $1000 ; starting address of the elements
array dw 320, 333, 321, 420, 500, 550, 620, 700, 400, 300
sum rmb 4 ; array sum
i rmb 1 ; array index
org $1500 ; starting addr of program
ldaa #0
staa sum ; initialize sum to 0
staa sum ;
staa sum ;
staa sum ;
staa i ; initialize loop counter zero
Example 8 : Loops Contd..

```
loop      ldab  i      ;B ← i
            cmpb  #N      ;is i=N?
            beq    done    ;if i=N then branch to label ‘done’
            ldx    #array  ;use index register X as pointer to
                            the array , X=$1000
            abx     ;X ← X+B = X+1, to compute the
                                ;addr of current element
            ldd     0,x      ;place array[i] in D
            addd   sum+2    ;D ← D+sum
            std     sum+2
            ldaa   #0
            adca   sum+1    ;propagating carry to msb
            staa   sum+1
            inc     i        ;moving to next element
            inc     I
            bra     loop
done      swi
end
```
Example 9: Bit Condition Branch Instructions

Write a program to count the number of elements that are divisible by 4 in an array of N 8-bit numbers.

Numbers divisible by 4 have the least significant two bits to be 00.

N equ 10
org $1000 ; starting address of the 1st element in the array
array db 1,2,3,4,5,6,7,8,9,10
total rmb 1 ; variable counting no. of elements divisible by 4
org $1500 ; starting address of the program
clr total ; initializing counter to 0
ldx #array ; loading x with the address of the array, X= $1000
ldab #N ; B is used as the loop counter
loop brclr 0,x,$03,yes

Operand at the memory location pointed by 0,x → 000000100
Operand provided by the mask in the instr → 00000011
AND result = 0
Execution will branch to the location specified in the label (yes)
Example 9: Bit Condition Branch Instructions

Write a program to count the number of elements that are divisible by 4 in an array of N 8-bit numbers

Numbers divisible by 4 have the least significant two bits to be 00

<table>
<thead>
<tr>
<th>N</th>
<th>equ</th>
<th>10</th>
<th>;starting address of the 1st element in the array</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>$1000</td>
<td>;variable counting no. of elements divisible by 4</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>db</td>
<td>1,2,3,4,5,6,7,8,9,10</td>
<td>;starting address of the program</td>
</tr>
<tr>
<td>total</td>
<td>rmb</td>
<td>1</td>
<td>;initializing counter to 0</td>
</tr>
<tr>
<td>org</td>
<td>$1500</td>
<td>;loading x with the address of the array, X= $1000</td>
<td></td>
</tr>
<tr>
<td>clr</td>
<td>total</td>
<td>;B is used as the loop counter</td>
<td></td>
</tr>
<tr>
<td>ldx</td>
<td>#array</td>
<td>;loading x with the address of the array, X= $1000</td>
<td></td>
</tr>
<tr>
<td>ldab</td>
<td>#N</td>
<td>;B is used as the loop counter</td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>brc</td>
<td>0,x,$03,yes</td>
<td>;add 1 to total as the number is divisible by 4</td>
</tr>
<tr>
<td>bra</td>
<td>chkend</td>
<td>;move the array pointer</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>inc</td>
<td>total</td>
<td></td>
</tr>
<tr>
<td>chkend</td>
<td>inx</td>
<td>b,loop</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 10: Bit Condition Branch Instructions

Write a program to count the number of elements that whose bit 1,4,7 are 1’s using brset.

```
N equ 10
org $1000 ; starting address of the 1st element in the array
array db 1,2,3,4,5,6,7,8,9,10
total rmb 1 ; variable counting no. of elements having the required pattern
org $1500 ; starting address of the program
clr total ; initializing counter to 0
ldx #array ; loading x with the address of the array, X = $1000
ldab #N ; B is used as the loop counter
loop brclr 0,x,$49,yes

Operand at the memory location pointed by 0,x → 11010110 → Inv → 00101001
Operand provided by the mask in the instr → 10010010
AND result = 0
Execution will branch to the location specified in the label (yes)
```

Mask = $49
### Example 10: Bit Condition Branch Instructions

Write a program to count the number of elements that whose bit 1,4,7 are 1’s using `brset`.

<table>
<thead>
<tr>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001010</td>
</tr>
</tbody>
</table>

```
N equ 10
org $1000 ; starting address of the 1st element in the array
array db 1,2,3,4,5,6,7,8,9,10
total rmb 1 ; variable counting no. of elements with the pattern
org $1500 ; starting address of the program
clr total ; initializing counter to 0
ldx #array ; loading x with the address of the array, X= $1000
ldab #N ; B is used as the loop counter
loop brset 0,x,$49,yes
bra chkend
yes inc total ; add 1 to total as the number has the required pattern
chkend inx ; move the array pointer
dbne b,loop
end
```
Example 11: Shift

Write a program to count the number of 1’s contained in the memory locations $2000$-$2001$ and save the result at memory location $1000$

```
org $2000
word: dc.w $2355 ;number to be worked on
one_ct rmb 1
lp_ct rmb 1
org $1500 ;starting addr of program
clr one_ct ;initializing to 0
movb #16,lp_ct ;initializing to 16
ldd word ;place 16 bit number in D
again lsrd ;shift right by 1 place
bcc chkend ;branch if lsb is a 0
inc one_ct ;in case of C=1
chkend dec lp_ct
bne again ;chk to see if we tested all ;16 bits
end
```

Loop count =16, Zero cnt =0;

D ← m[$2000]: m[$2001]

Shift D right by 1

If C=1

Zero count++

Dec loop cnt

If loop cnt = 16

END
Example 12: Boolean Logic Instructions

Write a sequence of instructions to clear the lower 4 pins of the I/O port located at $82 using AND

```assembly
ldaa $82 ;loading the contents of the mem location $82 into A
anda #$F0 ;clearing lower 4 bits in A
staa $82 ;A → m[$82]
```

<table>
<thead>
<tr>
<th>m[$82]</th>
<th>Mask</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>=10101010</td>
<td>=11110000</td>
<td>=10100000</td>
</tr>
</tbody>
</table>
Example 13: Boolean Logic Instructions

Write a sequence of instructions to set the bit 7 of the I/O port located at $82 using OR

```plaintext
ldaa $82 ;loading the contents of the mem location $82 into A
ora #$80 ;sets bit 7 in A
staa $82 ;A → m[$82]
```

```
| m[$82]  | =00101010 |
|-----------------|
| Mask = $80 = 10000000 |
| OR = 10101010 |
```
Example 14: Boolean Logic Instructions

Write a sequence of instructions to toggle the upper four bits of the I/O port at $82

ldaa $82 ;loading the contents of the mem location $82 into A
eora #$F0 ;toggles upper 4 bits in A
staa $82 ;A→m[$82]

------------------------------------------------------------------------------------------------------------------------
m[$82] =10101010
Mask=$F0=11110000
XOR =01011010
Example 15: Bit Test & Manipulate

Write a sequence of instructions to clear the upper four bits at $82

```
bclr $82,$F0
```

Write a sequence of instructions to set the upper four bits at $82

```
bset $82,$F0
```

Write a sequence of instructions to test the upper four bits at $82

```
ldaa $82
bita #$F0
```
Chapter Review

- Assembly Language Program Structure:
  - Label, operation, operand, comment
- Directives: end, org, db, ds, fill…
- Flow chart
- Arithmetic
- Loops, branch instructions
- Shift and rotate
- Boolean logic
- Bit test and manipulate
- Program execution time
Now, you should be able to:

- Allocate memory blocks, define constants, and create a message using assembler directives
- Write assembly programs to perform simple arithmetic operations
- Write loops to perform repetitive operations
- Use loops to create time delays
- Use boolean and bit manipulation instructions to perform bit field operations.