ECE3120: Computer Systems
Hardware & Software Development Tools

Manjeera Jeedigunta
http://blogs.cae.tntech.edu/msjeedigun21
Email: msjeedigun21@tntech.edu
Tel: 931-372-6181, Prescott Hall 120
Using the D-Bug12 Commands

- **BF <StartAddress> <EndAddress> [Data]**
  - Fill a block of memory locations with the value of `<Data>`.
    - To fill the memory locations from $1000$ to $1FFF$ with $0$, enter the following command:
      >bf 1000 1FFF 0

- **MD <StartAddress> [<EndAddress>]**
  - Display memory contents from `<StartAddress>` to `<EndAddress>`.
    - 16 bytes are displayed on each line.
    - Only one line is displayed if the `EndAddress` is not specified.
MDW  <StartAddress>  [<EndAddress>]

>md 1000
1000  AA 85 06 0C - D7 98 9A 61 - DF BE BC E9 - 03 AE D0 3D       a........=

>md 1000 1020
1000  AA 85 06 0C - D7 98 9A 61 - DF BE BC E9 - 03 AE D0 3D       a........=
1010  75 DA DF 39 - 3F 34 BD A9 - 2A CA FA DB - AC DA 18 97    u..9?4..*......
1020  4D 5B 48 BA - B2 F7 B6 1B - 92 99 E5 E4 - A5 E9 01 9F    M[H.............

>mdw 1000

1000  AA85 060C - D798 9A61 - DFBE BCE9 - 03AE D03D       a........=

>mdw 1000 1020

1000  AA85 060C - D798 9A61 - DFBE BCE9 - 03AE D03D       a........=
1010  75DA DF39 - 3F34 BDA9 - 2ACA FADB - ACDA 1897    u..9?4..*......
1020  4D5B 48BA - B2F7 B61B - 9299 E5E4 - A5E9 019F    M[H.............
MM <Address> [<Data>]

- Used to examine and modify the contents of memory locations one byte at a time.
- If the 8-bit data parameter is present on the command line, the byte at memory location
  - <Address> is replaced with <Data> and the command is terminated.
  - If no data is provided, then D-Bug12 enters the interactive memory modify mode.
  - In the interactive mode, each byte is displayed on a separate line following the address of data.
  - Single-character sub-commands are used for the modification and verification of memory contents in interactive mode.
  - The available sub-commands are as follows:
    - [<Data>] <CR> Optionally update current location and display the next location.
    - [<Data>] </> or <=> Optionally update current location and redisplay the same location.
    - [<Data>] <^> or <-> Optionally update current location and display the previous location.
    - [<Data>] <.> Optionally update current location and exit Memory Modify.
>mm 1000
1000 00
1001 00 FF
1002 00 ^
1001 FF
1002 00
1003 00 55 /
1003 55 .
>

**MMW <Address> [Data]**
- Allows the contents of memory to be examined and/or modified as 16-bit hex data.
- If the 16-bit data is present on the command line, the word at memory location <Address> is replaced with <Data> and the command is terminated.
- If no data is provided, then D-Bug12 enters the interactive memory modify mode.
- MMW supports the same set of sub-commands as does the MM command.
>mmw 1100
1100 00F0
1102 AA55 0008
1104 0000 ^
1102 0008 aabb
1104 0000
1106 0000 .
>
Move <StartAddress> <EndAddress> <DestAddress>
- The number of bytes moved is one more than <EndAddress> - <StartAddress>

>move 1000 10ff 1100
>
RD – register display
>rd
PP  PC  SP  X  Y  D = A:B  CCR = SXHI NZVC
38 1521 3C00 2014 0000 6E:14 1001 0100
xx:1521 9C42 CPD $0042
>
RM – register modification

>rm
PC=0000 1500
SP=0A00
IX=0000 0100
IY=0000
A=00
B=00 ff
CCR=90 d1
PC=1500 .
>

<RegisterName> <RegisterValue>
- Allow one to change the value of any CPU register.
- Each bit of the CCR register can be changed by specifying its name.
>pc 2000
   PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
2000  0A00  0100  0000     00:FF        1101 0001
>x 800
   PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
2000  0A00  0800  0000     00:FF        1101 0001
>c 0
   PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
2000  0A00  0800  0000     00:FF        1101 0000
>z 1
   PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
2000  0A00  0800  0000     00:FF        1101 0100
>d 2010
   PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
2000  0A00  0800  0000     20:10        1101 0100
>
Table 3.4 Condition code register bits

<table>
<thead>
<tr>
<th>CCR bit name</th>
<th>Description</th>
<th>Legal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>STOP enable</td>
<td>0 or 1</td>
</tr>
<tr>
<td>H</td>
<td>Half carry</td>
<td>0 or 1</td>
</tr>
<tr>
<td>N</td>
<td>Negative flag</td>
<td>0 or 1</td>
</tr>
<tr>
<td>Z</td>
<td>Zero flag</td>
<td>0 or 1</td>
</tr>
<tr>
<td>V</td>
<td>Two's complement over flg</td>
<td>0 or 1</td>
</tr>
<tr>
<td>C</td>
<td>Carry flag</td>
<td>0 or 1</td>
</tr>
<tr>
<td>IM</td>
<td>IRQ interrupt mask</td>
<td>0 or 1</td>
</tr>
<tr>
<td>XM</td>
<td>XIRQ interrupt mask</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

- Invokes the one-line assembler/disassembler.
- Allows memory contents to be viewed and altered using assembly language mnemonics.
- When displaying instructions, each instruction is displayed in its mnemonic form.
- The assembly/disassembly process can be terminated by a period.
- The one-line assembler displays the current instruction and allows the user to enter new instruction.
- User can skip the current instruction by pressing the **Enter** key.
The following example **displays** instruction starting from $2000:

```assembly
>asm 2000
2000  FC0800    LDD   $0800     >
2003  CD0900    LDY   #$0900    >
2006  CE000A    LDX   #$000A    >
2009  1810      IDIV             >
200B  CB30      ADDB  #$30     >
200D  6B44      STAB  4,Y       >
200F  B7C5      XGDX            >
2011  CE000A    LDX   #$000A   >.
```

The following example **enters** three instructions (in bold face) starting from $1500:

```assembly
>asm 1500
1500  FC0800    LDD   $0800
1503  F30802    ADDD  $0802
1506  7C0900    STD   $0900
1509  E78C      TST   12,SP    >.
```
BR [<Address> …] Setting or Examine Breakpoints

- A breakpoint halts the program execution when the CPU reaches the breakpoint address.
- When a breakpoint is encountered, the D-Bug12 monitor displays the contents of CPU registers and the instruction at the breakpoint (not executed yet).
- Breakpoints are set by typing the breakpoint command followed by one or more breakpoint addresses.
- Entering the breakpoint command without any breakpoint addresses will display all the currently set breakpoints.
- A maximum of ten user breakpoints may be set at one time.

```plaintext
>br 1020 1040 1050 ; set three breakpoints
Breakpoints: 1020 1040 1050
>br
Breakpoints: 1020 1040 1050
> ; display current breakpoints
```
NOBR [<Address> <Address>]

- Delete one or more previously defined breakpoints.
- All breakpoints will be deleted if no addresses are specified.

> br 2000 2010 2020 2040 2090 ; set four breakpoints
Breakpoints: 2000 2010 2020 2040 2090

> nbr 2000 2010 ; delete two breakpoints
Breakpoints: 2020 2040 2090

> nbr
All Breakpoints Removed

>
**G [Address]**

- Begin execution of user code at the specified address.
- If no address is specified, CPU starts execution of the instruction at the current PC address.

```
>g 1500
User Bkpt Encountered
PP  PC  SP  X  Y  D = A:B  CCR = SXHI NZVC
38 150C 3C00 7B48 0000 03:E8 1001 0001
xx:150C 911E CMPA $001E
>
```
GT <Address>

- Execute instruction until the given address and stop.
- User usually needs to specify where the program execution should start before issuing this command.

```
>pc 1500
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1500 3C00 1000 1002 00:00 1001 0101
xx:1500 CF1500 LDS #$1500
>gt 1540
Temporary Breakpoint Encountered
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1510 1500 1000 1002 1E:00 1001 0000
xx:1510 3B PSHD
>```

Copyright © 2006 Thomson Delmar Learning
T [<count>]

- Used to execute one or multiple instructions starting from the current PC address.
- As each program instruction is executed, the CPU register contents and the next instruction to be executed are displayed.
- Only one instruction will be executed when no count is specified.

```
>pc 1500
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1500 1500 1000 1002 1E:00
xx:1500 CF1500 LDS #$1500
>t
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1503 1500 1000 1002 1E:00
xx:1503 CE1000 LDX #$1000
>t 2
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1506 1500 1000 1002 1E:00
xx:1506 34 PSHX
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1507 14FE 1000 1002 1E:00
xx:1507 861E LDAA #$1E
>```
CALL [<Address>]

- Used to execute a subroutine and returns to the D-Bug12 monitor program.
- All CPU registers contain the values at the time the final RTS instruction was executed, with the exception of the program counter.
- The program counter contains the starting address of the subroutine when returning from the subroutine.

>call 1600
Subroutine Call Returned
pp PC SP X Y D = A:B CCR = SXHI NZVC
38 1600 0A00 0032 0900 00:31 1001 0000
xx:1600 FC1000 LDD $1000
>
Tips for Assembly Program Debugging

• Syntax errors
  – Misspelling of instruction mnemonics
    • Starting instruction mnemonic at column 1. The mnemonic is treated as a label whereas the operands are treated as mnemonic.
  – Missing operands
    • Will be highlighted by the assembler and are easy to fix.

• Logic errors
  – Using extended (or direct) mode instead of immediate mode
    • A program with this type of addressing mode error is on the next page.
N equ 20 ; array count
org $1000
array dc.b 2,4,6,8,10,12,14,16,18,20
   dc.b 22,24,26,28,30,32,34,36,38,40
sum  ds.w 1

org $1500
ldx array ; place the starting address of array in X
movw 0,sum ; initialize sum to 0
ldy N ; initialize loop count to N
loop ldab 1,x+ ; place one number in B and move array pointer
   sex B,D ; sign-extend the 8-bit number to 16-bit
   addd sum ; add to sum
   std sum ; update the sum
   dbne y,loop ; add all numbers to sum yet?
   swi ; return to monitor
end

- Assemble and download this program onto the demo board.

>load
....
done
>
• Use the **asm** command to make sure that the program is downloaded correctly.

```
>asm 1500
xx:1500  FE1000  LDX  $1000   >
xx:1503  180400001014  MOVW  $0000,$1014  >
xx:1509  DD14  LDY  $0014  >
xx:150B  E630  LDAB  1,X+  >
xx:150D  B714  SEX  B,D  >
xx:150F  F31014  ADDD  $1014  >
xx:1512  7C1014  STD  $1014  >
xx:1515  0436F3  DBNE  Y,$150B  >
xx:1518  3F  SWI  >.
```

• Make sure that program data is downloaded correctly. Use the **md** command:

```
>md 1000 1010
1000 02 04 06 08 - 0A 0C 0E 10 - 12 14 16 18 - 1A 1C 1E 20  ............
1010 22 24 26 28 - 00 00 B9 A9 - 2A CA FA DB - AC DA 18 97 "$&(....*........
```
Run the Program

>g 1500
User Bkpt Encountered
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1519 3C00 0213 0000 FF:07 1001 1000
xx:1519 88F4 EORA #$F4
>
Exam the execution result – incorrect!!
>md 1010
1010 22 24 26 28 - FF 07 B9 A9 - 2A CA FA DB - AC DA 18 97
>
• The program is short.
• Errors can be found by tracing.
• Set PC to the start of the program (at $1500)

>pc 1500
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 1500 3C00 0213 0000 FF:07 1001 1000
xx:1500 FE1000 LDX $1000
>

Trace One Instruction at a Time

• The executed instruction is “ldx $1000” which should place the
  start address of the array in X.
• The instruction trace result shows that X receives $0204, not
  $1000.
• This is due to addressing mode error.
• Change the instruction to \texttt{ldx \#$1000} and rerun the program.
• Reload the program and trace the program.
• Trace two instructions this time.
• We expect the variable `sum` (at $1014$ and $1015$) to receive $0000$. But it didn’t.
• The error is again caused by incorrect use of the addressing mode.
• The `movm 0,sum` instruction copies the contents of memory location 0 to `sum`.
• Change the second instruction to `movw #0,sum`. Rerun the program and examine the memory contents.
• It is still incorrect !!
>load
*
>g 1500
User Bkpt Encountered
PP  PC  SP    X  Y  D = A:B  CCR = SXHI NZVC
38 1519 3C00 100F 0000 00:F0 1001 0000
xx:1519 88F4  EORA  #$F4
>md 1010
1010 22 24 26 28 - 00 F0 B9 A9 - 2A CA FA DB - AC DA 18 97
>

• Trace the program up to the third instruction:
The program intends to load 20 into Y with the third instruction and expect Y to be set to 20. But Y did not get 20. It receives 0F instead.

• This is due to the incorrect use of the addressing mode.
• Change the instruction to `ldy #20` and rerun the program.
>g 1500
User Bkpt Encountered
PP PC SP X Y D = A:B CCR = SXHI NZVC
38 151A 3C00 1014 0000 01:A4 1001 0000
xx:151A F421BD ANDB $21BD
>md 1010
1010 22 24 26 28 - 01 A4 B9 A9 - 2A CA FA DB - AC DA 18 97
>

- After this correction, sum receives the correct value $1A4 (420).
Mismatch of Operand Size

**Example Program** – Finding the sum of elements of an array

N equ 20 ; array count
org $1000
array dc.b 2,4,6,8,10,12,14,16,18,20
dc.b 22,24,26,28,30,32,34,36,38,40
sum ds.w 1

org $1500
ldx #array ; place the starting address of array in X
movw #0,sum ; initialize sum to 0
ldy #N ; initialize loop count to N
loop ldd 1,x+ ; place one number in D and move array pointer
add sum ; add to sum
std sum ; update the sum
dbne y,loop ; add all numbers to sum yet?
swi ; return to monitor
end
•The value of **sum** is incorrect after running the program:

```
>md 1010
1010 22 24 26 28 - A6 1F B9 A9 - 2A CA FA DB - AC DA 18 97
```

This program can be debugged by tracing:

```
>pc 1500
PP  PC   SP  X     Y    D = A:B   CCR = SXHI NZVC
38 1500  3C00  1014  0000     A6:1F        1001 1000
xx:1500  CE1000   LDX #$1000
>t
PP  PC   SP  X     Y    D = A:B   CCR = SXHI NZVC
38 1503  3C00  1000  0000     A6:1F        1001 0000
xx:1503  180300001014  MOVW #$0000,$1014
>t
PP  PC   SP  X     Y    D = A:B   CCR = SXHI NZVC
38 1509  3C00  1000  0000     A6:1F        1001 0000
xx:1509  CD0014   LDY #$0014
>t
PP  PC   SP  X     Y    D = A:B   CCR = SXHI NZVC
38 150C  3C00  1000  0014     A6:1F        1001 0000
xx:150C  EC30   LDD 1,X+
```
The 4th instruction should place the value 2 in D rather than $0204. This is due to the incorrect use of the instruction of `ldd 1,x+`. This instruction should be replaced by the following two instructions:

```
ldab 1,x+
clra
```

• Other logic errors:
  • **Inappropriate Use of Index Addressing Mode**
  • Indexed addressing mode is often used to step through array elements.
  • After accessing each element, the index register must be incremented or decremented.
  • Program execution can’t be correct if index register is incremented or decremented incorrectly.
  • This error can be found after performing computation in the first one or two elements by program tracing.