

Introduction to Microcontrollers II

brset, brclr

Indexed Addressing Example

μp Laboratory #2

BUFFALO

Assembling Code

EECE 143 Digital Design Project

Purpose: To allow students to design their own digital project in order to demonstrate the utilization of digital design concepts.

Examples: Digital Alarm Clock, Traffic Light Simulation, Soda pop machine controller
Scoreboard Display with timers

Design Criteria: If required, all clock sources must be designed and built by students.
You may use IC's available in the Open Lab.
You may use discrete LS or HC ICs as well as GALs.

Grading:

| | | |
|--|------------|---|
| Design Proposal | 10 | DUE: Monday, November 5th in class |
| <i>(Preliminary design diagrams, flowcharts)</i> | | |
| Design Project | 30 | Presentations & Reports: Thursday, Dec 6th |
| <i>(Neatness of design, design complexity Making it work)</i> | | |
| Presentations | 60 | |
| <i>(Visual Aids, Ability to communicate design, Peer, TA and Teacher Evals)</i> | | |
| Final Report (Writeup) | 100 | |
| <i>(Purpose, Preparation, Experiment Procedure, Design Description, Schematics, Flowcharts, Code, Future Design Improvements and Considerations)</i> | | |
| Total | 200 | |

Digital Design Projects

- Digital Alarm Clock
- Digital Temperature Buffer/Storage
- Soda Pop Machine with Password for Free Soda
- Car Turning Signal Light Controller
- Electric Door Lock Control Circuit
- Traffic Light Signal Controllers
- Digital Tic-Tac-Toe
- Digital Slot Machine
- Digital Kitchen Timer
- Sports Scoreboard with Timer
- Digital Sampler and Playback

Design Project Presentation

- Thursday December 6
- 5-10 minutes to present your project and demonstrate
- You will be graded on
 - How well you communicate your design
 - How well you answer questions
 - Audio/visual aids (Powerpoint presentations, poster boards...)

Design Project Written Report

- Due at time of presentation
- Typed
- Should include the following
 - Purpose
 - Design Description: A module by module description of your design.
 - Equations used in your design. Truth Tables...
 - COMPLETE schematic diagrams, block diagrams
 - Flow charts and Code (if applicable)
 - Testing Procedures: Give a step by step account of how you test to verify your circuit works.
 - Future Design Improvements and Conclusion

Design Project Grade Breakdown

| | |
|-------------------|-----|
| • Design Proposal | 10 |
| • Design Project | 30 |
| • Presentations | 60 |
| • Written Report | 100 |
| Total | 200 |

A Look at Appendix A of the HC11 Reference Manual

Load Accumulator

LDA

Load Accumulator

LDA**Operation:** $AccX \leftarrow (M)$ **Description:** Loads the contents of memory into the 8-bit accumulator. The condition codes are set according to the data.**Condition Codes and Boolean Formulae:**

| S | X | H | I | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| - | - | - | - | | | 0 | - |

N R7
Set if MSB of result is set; cleared otherwise

Z $R7' \bullet R6' \bullet R5' \bullet R4' \bullet R3' \bullet R2' \bullet R1' \bullet R0'$
Set if result is \$00; cleared otherwise

V 0
cleared

Source Form: LDAA (opr); LDAB (opr)

Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

| Cycle | LDAA (IMM) | | | LDAA (DIR) | | | LDAA (EXT) | | | LDAA (IND,X) | | | LDAA (IND,Y) | | |
|-------|------------|------|------|------------|--------|------|------------|--------|------|--------------|--------|------|--------------|--------|------|
| | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* |
| 1 | OP | 86 | 1 | OP | 96 | 1 | OP | B6 | 1 | OP | A6 | 1 | OP | 18 | 1 |
| 2 | OP+1 | ii | 1 | OP+1 | ii | 1 | OP+1 | hh | 1 | OP+1 | ff | 1 | OP+1 | A6 | 1 |
| 3 | | | | 00dd | (00dd) | 1 | OP+2 | ll | 1 | FFFF | -- | 1 | OP+2 | ff | 1 |
| 4 | | | | | | | hhll | (hhll) | 1 | X+ff | (X+ff) | 1 | FFFF | -- | 1 |
| 5 | | | | | | | | | | | | | Y+ff | (Y+ff) | 1 |

| Cycle | LDAB (IMM) | | | LDAB (DIR) | | | LDAB (EXT) | | | LDAB (IND,X) | | | LDAB (IND,Y) | | |
|-------|------------|------|------|------------|--------|------|------------|--------|------|--------------|--------|------|--------------|--------|------|
| | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* | Addr | Data | R/W* |
| 1 | OP | C6 | 1 | OP | D6 | 1 | OP | F6 | 1 | OP | E6 | 1 | OP | 18 | 1 |
| 2 | OP+1 | ii | 1 | OP+1 | ii | 1 | OP+1 | hh | 1 | OP+1 | ff | 1 | OP+1 | E6 | 1 |
| 3 | | | | 00dd | (00dd) | 1 | OP+2 | ll | 1 | FFFF | -- | 1 | OP+2 | ff | 1 |
| 4 | | | | | | | hhll | (hhll) | 1 | X+ff | (X+ff) | 1 | FFFF | -- | 1 |
| 5 | | | | | | | | | | | | | Y+ff | (Y+ff) | 1 |

brset command

- Branch if Bit(s) Set:

Performs the logical AND of location M inverted and the mask supplied with the instruction, then branches if the result is zero (only if all bits corresponding to ones in the mask byte are ones in the tested byte)

brset \$1004 %00001111 GOHERE

Will branch to location GOHERE if 4 LSBs of PortE are all 1's

brclr command

- Branch if Bit(s) Clear:

Performs the logical AND of location M and the mask supplied with the instruction, then branches if the result is zero (only if all bits corresponding to ones in the mask byte are zeros in the tested byte)

brset \$1004 %00001111 GOHERE

Will branch to location GOHERE if 4 LSBs of PortE are all 0's

```

0001      * COUNT_BR.A11
0002      *      Count pulses at an input.
0003      *      Two digit BCD output.
0004      *      Bruce Hoepfner   11/10/92
0005      *                               06/19/94 brset, brclr
0006      * NOTE: modifications not tested
0007      * NOTE: bset, bclr, brset, brclr instructions
0008      *      work for direct (page 0) or indexed mode
0009
0010      *      Bounceless input at bit0 of PORTE
0011      *      Output to PORTB
0012      * Constants - Hardware dependent
0013 1000      REGBASE      equ $1000
0014 0000      _PORTA      equ $00
0015 0004      _PORTB      equ $04
0016 000a      _PORTE      equ $0a
0017 0001      BIT0 equ %00000001
0018      *****
0019
0020
0021 c000      org $C000      ;origin in user RAM
0022      * Initialize
0023 c000 4f      MAIN      clra
0024 c001 ce 10 00      ldx #REGBASE
0025 c004 a7 04      staa _PORTB,x      ;Initialize Output
0026
0027      * Looking for a POSITIVE EDGE
0028      * First: Loop until input = 0 (loop while bit=1)
0029 c006 1e 0a 01 fc      WAIT0      brset _PORTE,x BIT0 WAIT0
0030
0031      * above one line replaces following three,
0032      * and it is more clear
0033      * WAIT0 ldab $100a      ;read input
0034      *      andb #$01      ;mask off 7 MSBs
0035      *      bne WAIT0
0036

```

Example:

count_br.lst

This program uses the brset
brclr commands which
check to see if a bit or bits
are set or cleared.

```
0037          * Then: Loop until input = 1 (loop while bit=0)
0038 c00a 1f 0a 01 fc    WAIT1      brclr _PORTE,x BIT0 WAIT1
0039
0040          * above one line replaces following three,
0041          * and it is more clear
0042          *WAIT1ldab $100A      ;read input
0043          *      andb #$01      ;mask off 7 msbs
0044          *      beq WAIT1
0045
0046          * Only get here if after a +edge at specified bit
0047
0048          *Increment Count (Use adda so daa works)
0049 c00e 8b 01          adda #$01      ;increment AccA
0050 c010 19            daa          ;adjust for BCD
0051
0052 c011 a7 04          staa _PORTB,x    ;write to PORTB
0053 c013 7e c0 06      jmp WAIT0
```

When you compare this code with that of count.a11
you will see it is a lot shorter and easier to understand

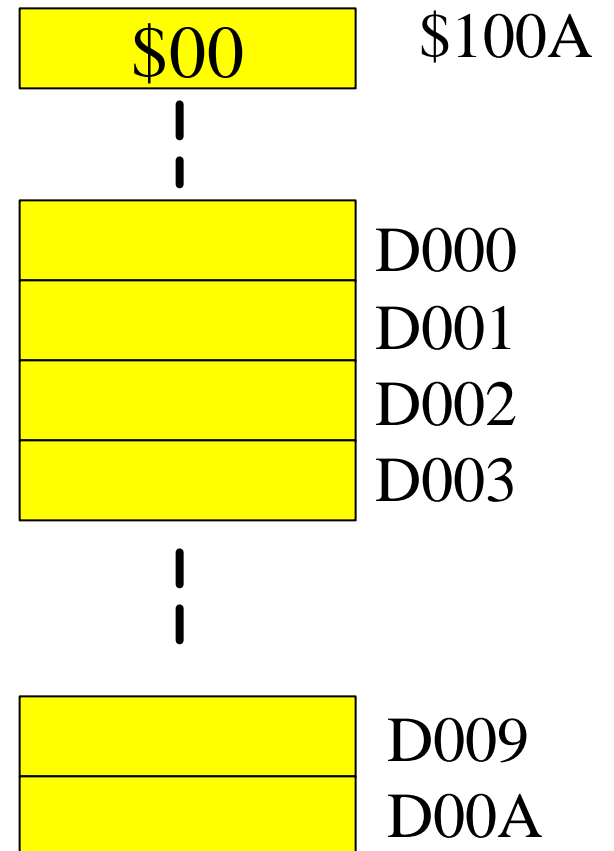
Using Indexed Addressing in Loops

```
PORTE equ $100A    ;Set definition for PORTE
BUFFALO equ $e00a ;Set definition for BUFFALO
    org $c000      ;Store program at $c000
    ldx #$D000     ;load initial value of x register
READ ldaa PORTE    ; load value into AccA
    staa 0,x       ; store value of AccA into memory location
    inx           ;Increment Register x
    cmpx #$D00A   ;Is x Register = $D00A?
    blt READ      ;If less than get another value
    jmp BUFFALO   ; If it is, end program
end
```

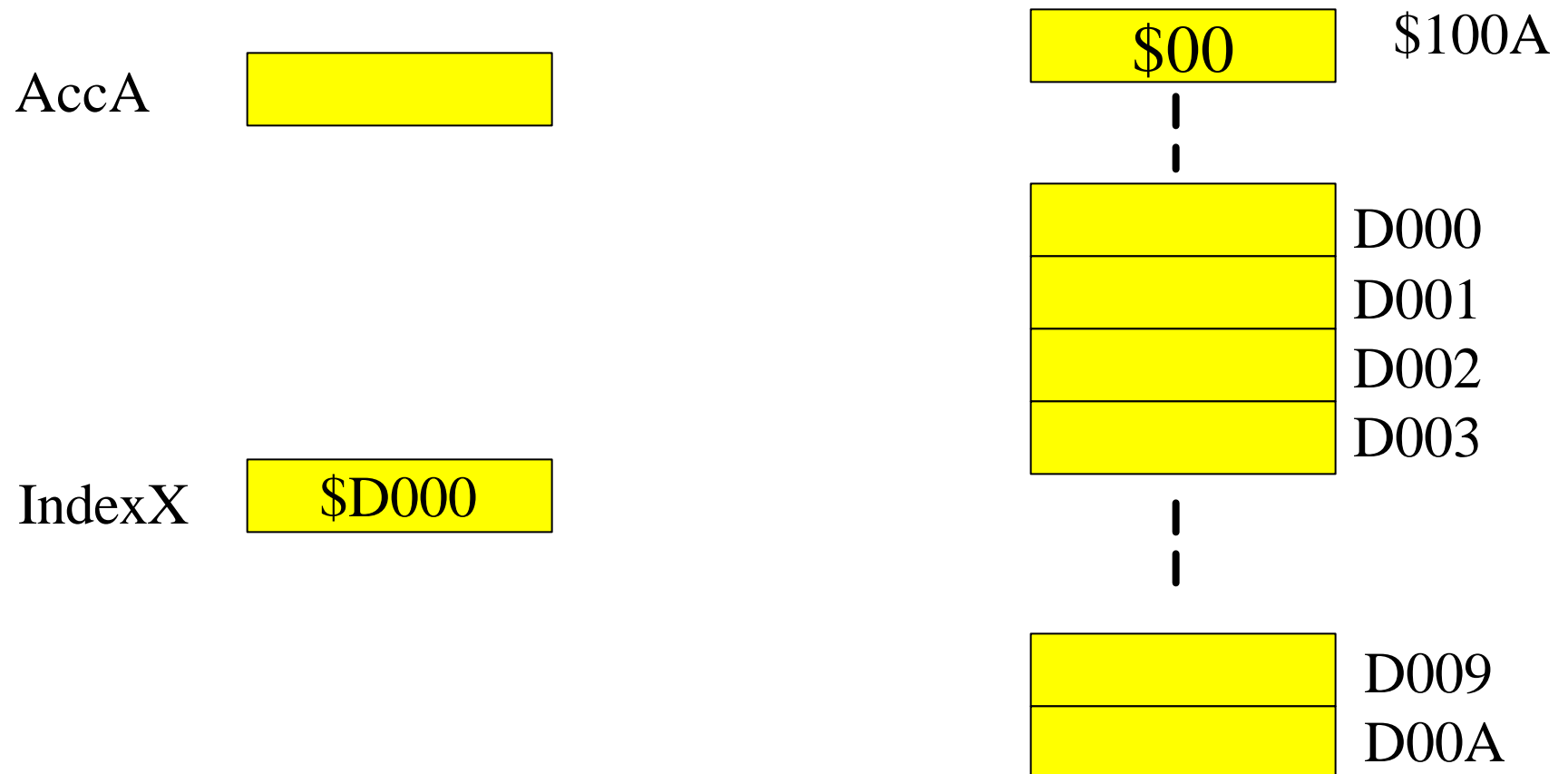
AccA



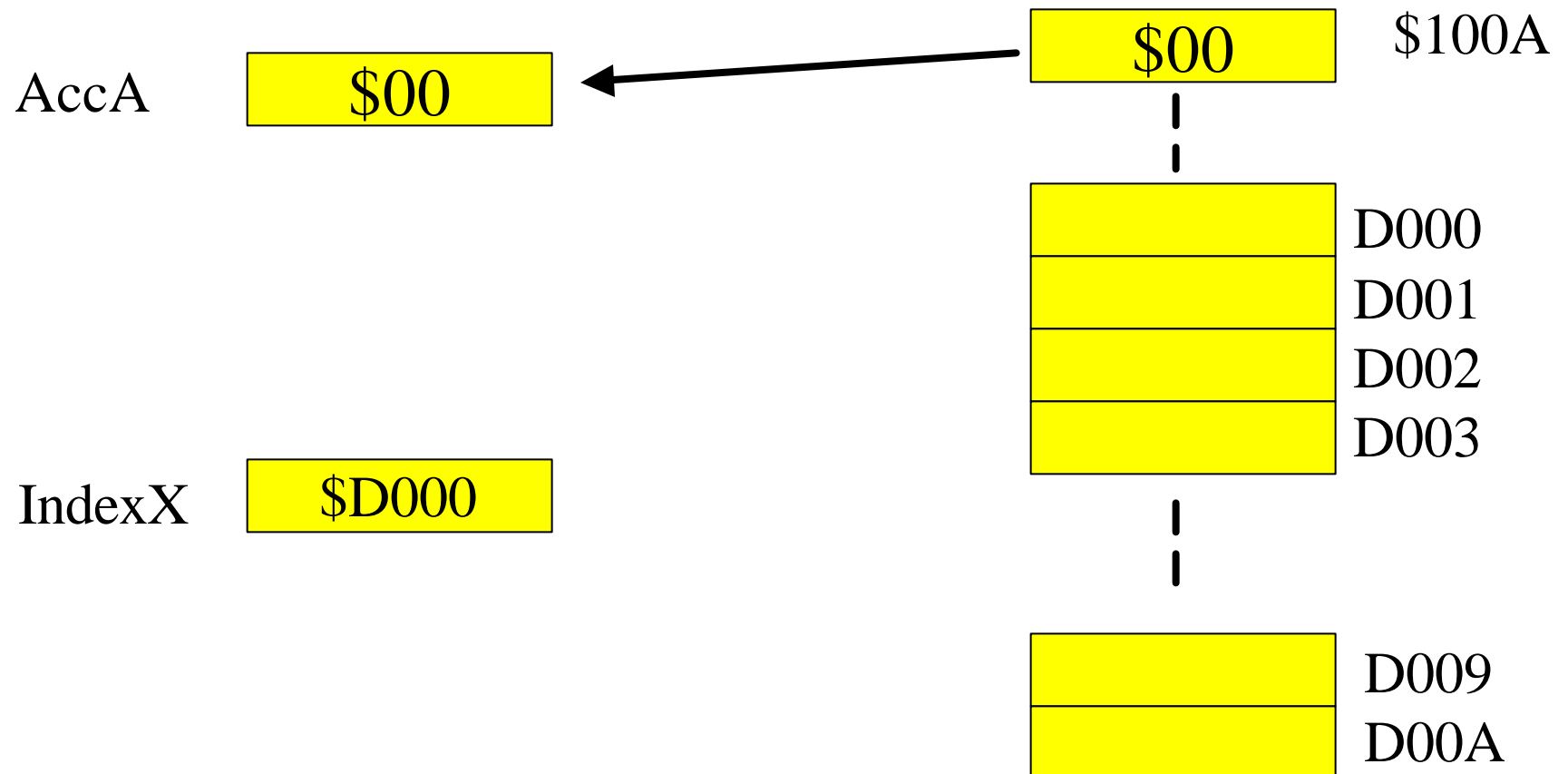
IndexX



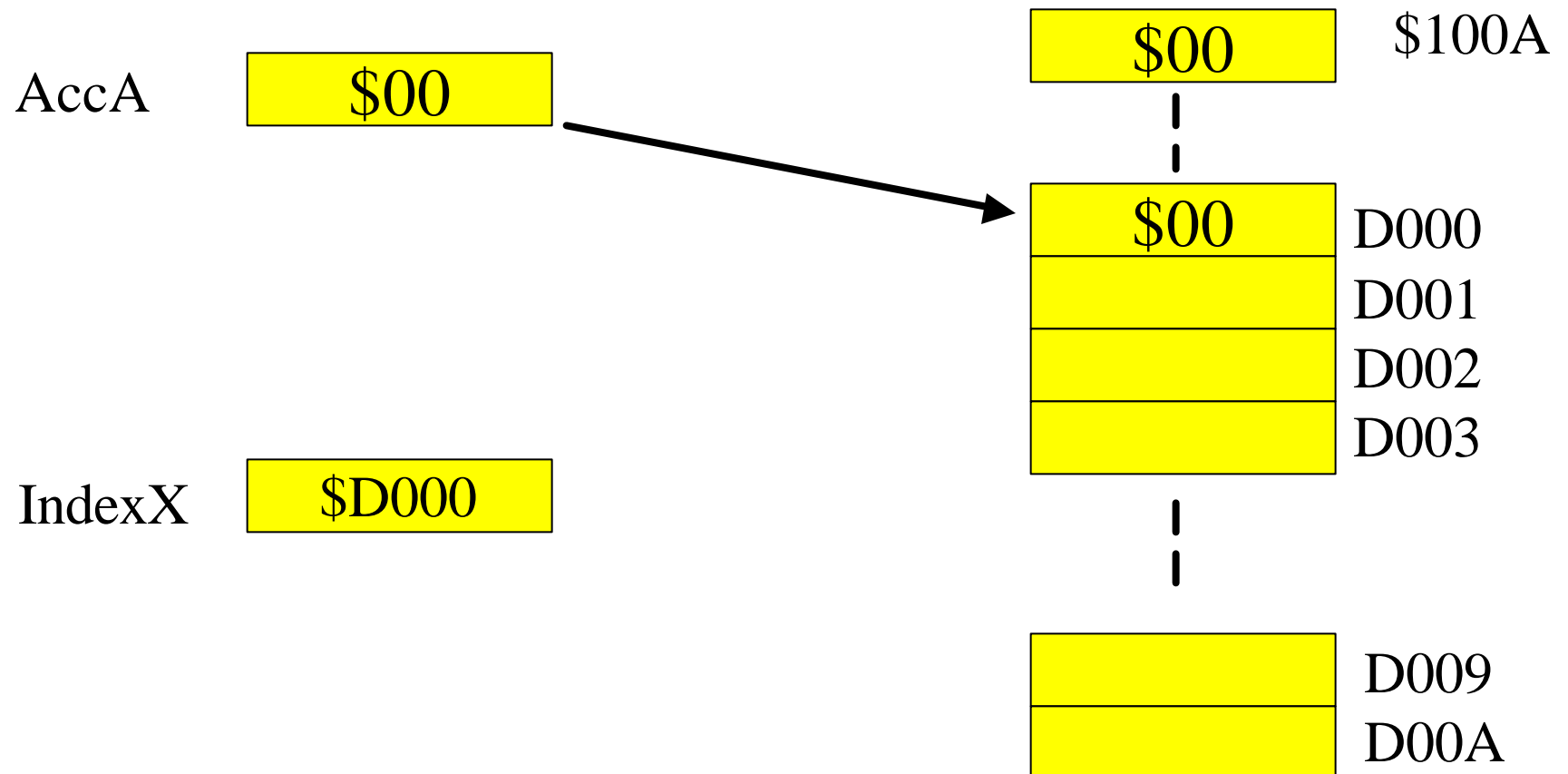
**ldx #\$D000 : Loads Index Register X with \$D000
immediate mode**



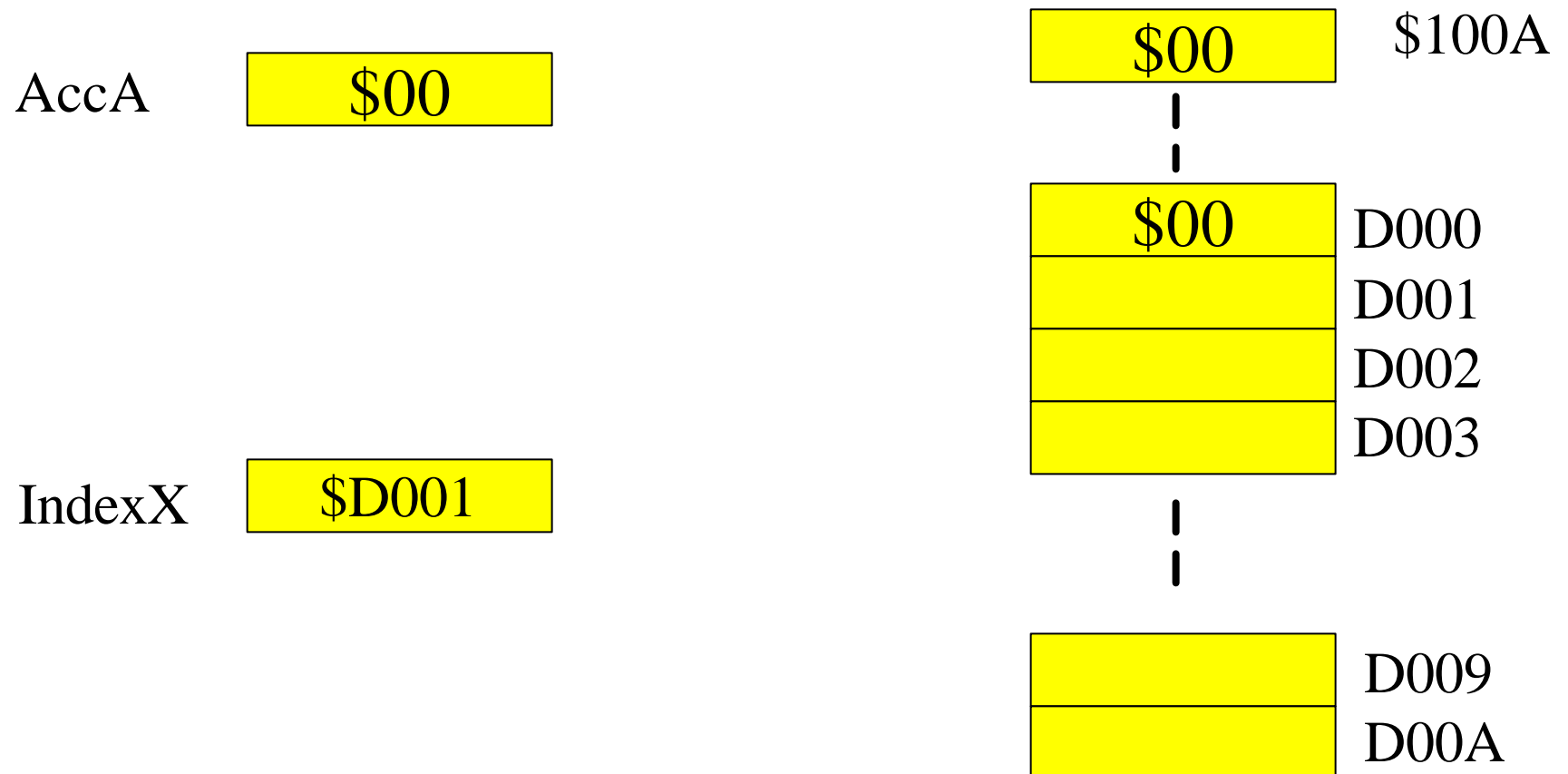
**ldaa PORTE : loads Acca with the contents of
PORTE (\$100A)**



**staa 0,x ;Puts the contents of AccA into the
memory location [D000 + 0]**



inx : increments the Index register X



**cmpx #\$D00A : compares the contents of index
register X with \$D00A immediate mode**

AccA

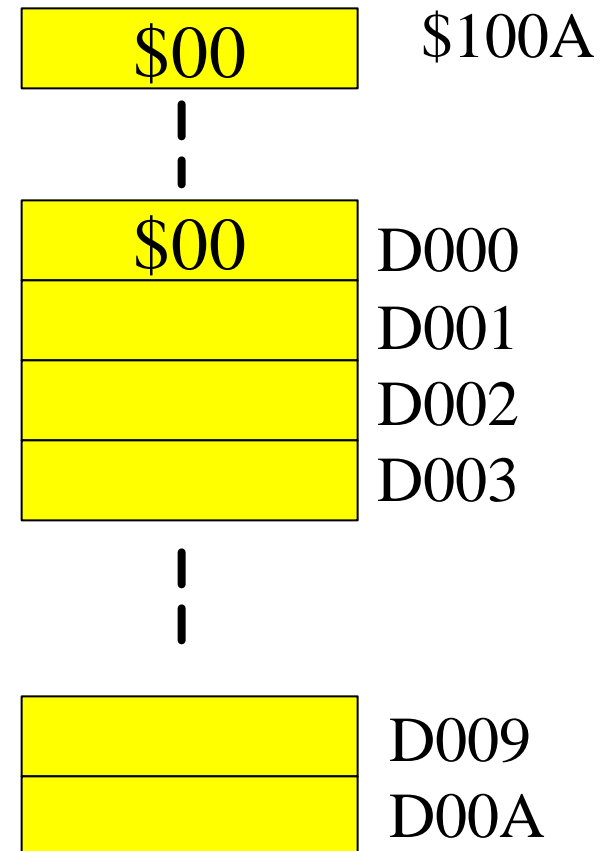
| |
|------|
| \$00 |
|------|

IndexX

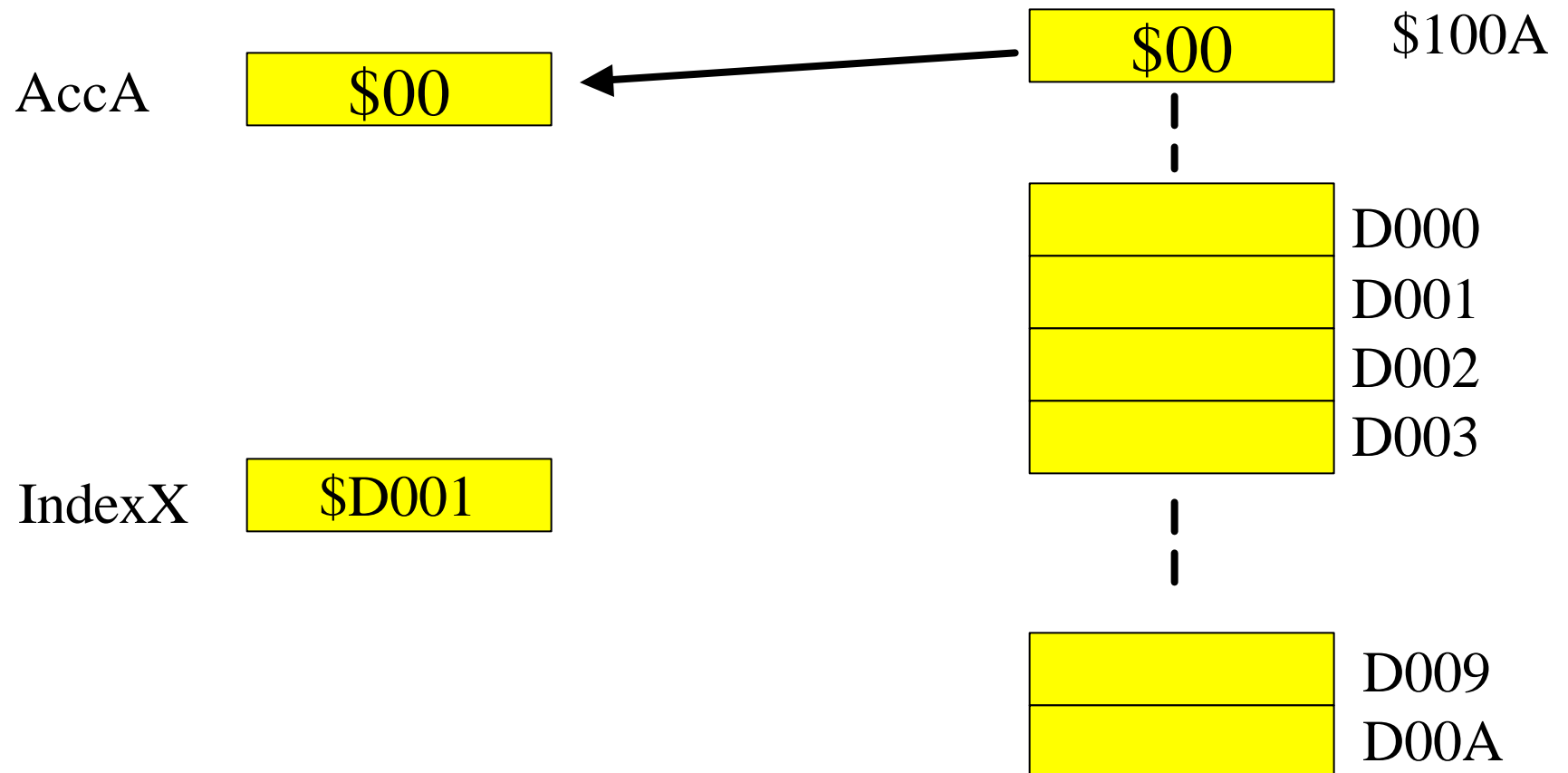
| |
|--------|
| \$D001 |
|--------|

= \$D00A?

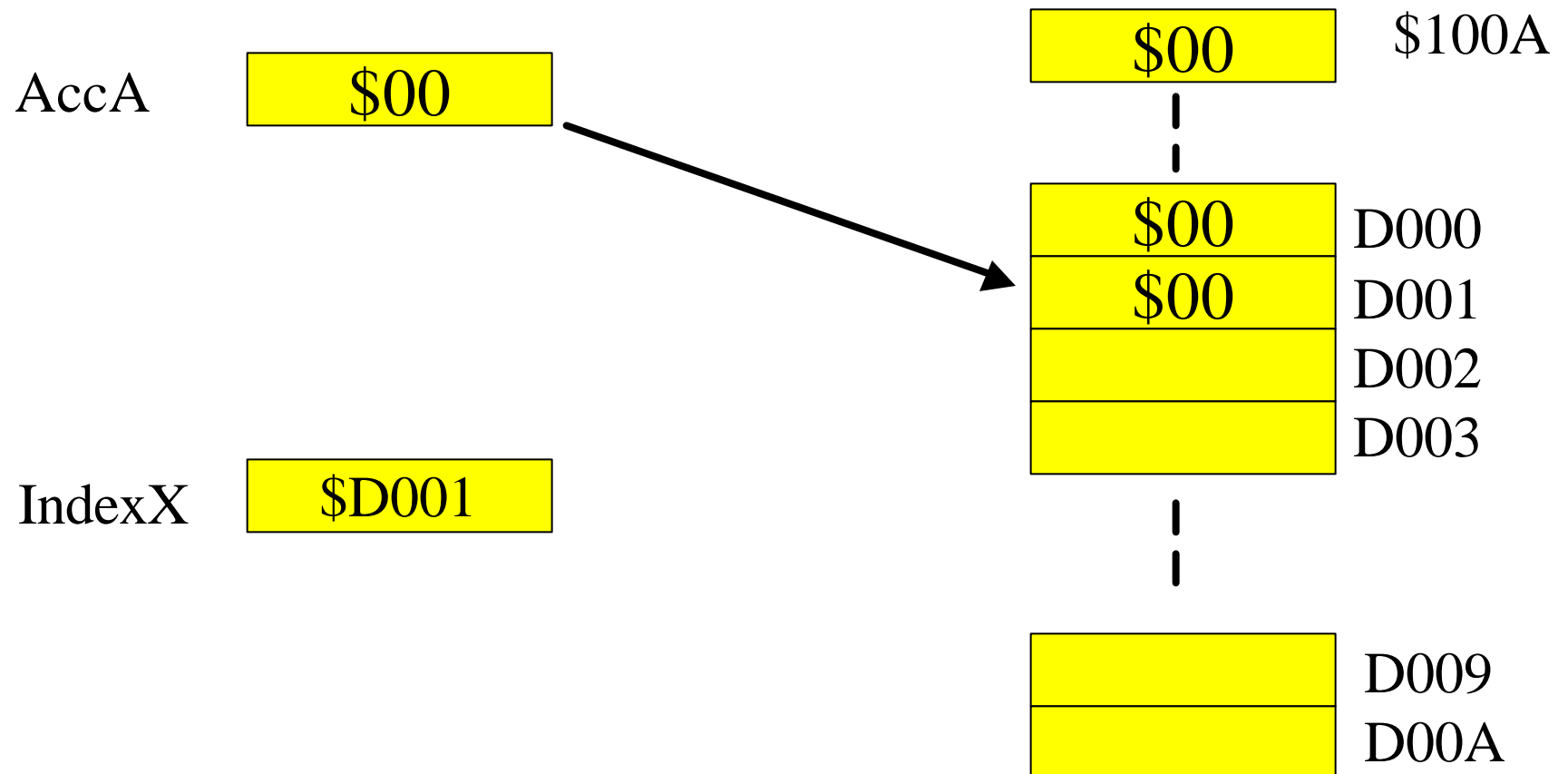
If less than, it branches back to
READ, otherwise jmp BUFFALO



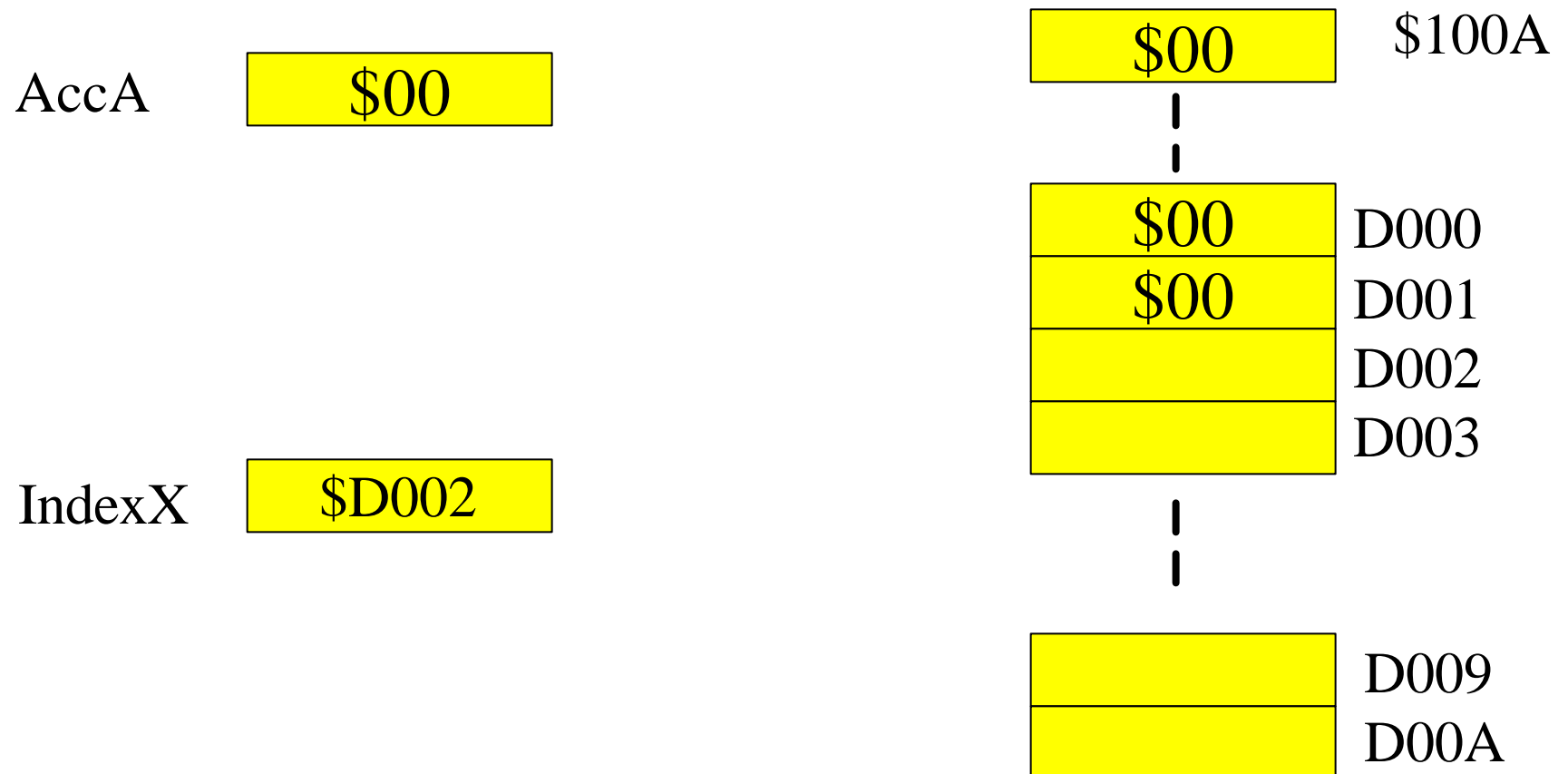
**ldaa PORTE : loads Acca with the contents of
PORTE (\$100A)**



**staa 0,x ;Puts the contents of AccA into the
memory location [D001 + 0]**



inx : increments the Index register X



**cmpx #\$D00A : compares the contents of index
register X with \$D00A immediate mode**

AccA

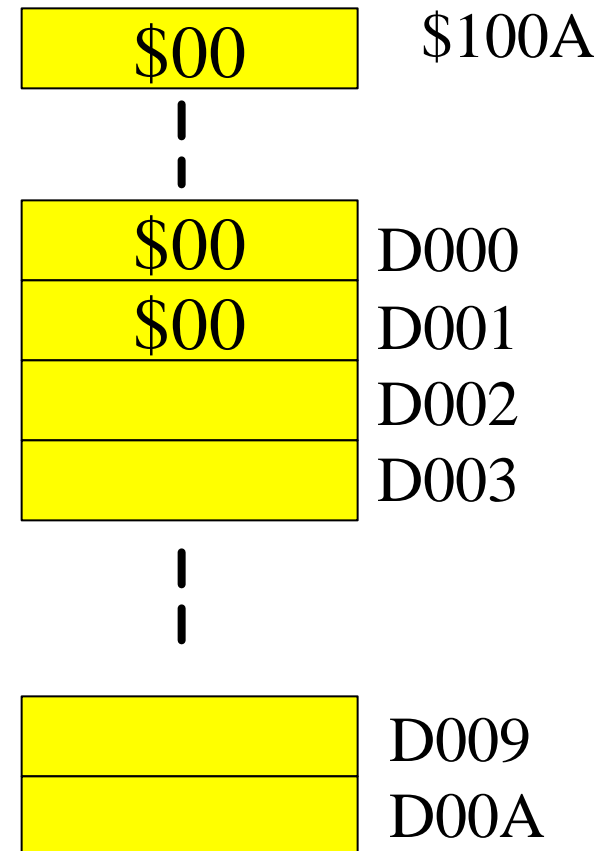
| |
|------|
| \$00 |
|------|

IndexX

| |
|--------|
| \$D001 |
|--------|

= \$D00A?

If less than, it branches back to
READ, otherwise jmp BUFFALO



Guide to A11 Files

```
PORTB equ $1004
```

```
    org $C000
```

```
MAIN clra
```

```
    staa PORTB
```

```
WAIT0 ldab $100a
```

```
    andb #$01
```

```
    bne WAIT0
```

```
WAIT1 LDAB $100a
```

```
    andb #$01
```

```
    beq WAIT1
```

```
    adda #$01
```

```
    daa
```

```
    staa $1004
```

```
    jmp WAIT0
```

;constant declarations should be put before the org statement

;org <address to store program>

;labels and constants should start in column 1, all other statements should be tabbed to the right

;Default is decimal, use \$ for HEX or % for binary

;Anything after a “;” or a * is a comment

AS11 Common Mistakes

see page 9-23

- Missing “#” Motorola defaults to direct or extended addressing mode.
- Using signed branches on unsigned data. AS11 has both signed and unsigned conditional branches. Unsigned (BHI BLO BHS BLS) Signed (BGT BLT BGE BLE)
- Flow charts drawn after code is written. Know what you want to do before you try to write the code.
- Missing a “\$”. AS11 defaults to decimal. NOTE: BUFFALO only allows HEX. No “\$” is needed when using the onboard assembler ASM.
- Loading a 16-bit value into an 8-bit location or vice versa.
- Improper ending for embedded code. Use (in)finite loop or jump to BUFFALO.
- On reads: high byte first, low byte second.
- Errors when pushing and pulling from stack.
- Memory usage
- high byte $\text{AccD} == \text{AccA}$, low byte $\text{AccD} == \text{AccB}$

Code Size and Execution Time

```

0001          * IndStor.a11
0002          * J. Chris Perez
0003          * Program uses indexed registering to loop
0004          * through a set of instructions
0005          * to increment value located
0006          * in AccA and store result in consecutive
0007          * Memory locations
0008
0009 c000          org $c000    ;Set location of program in memory
0010 c000 4f          clra          ;Clear Accumulator A
0011 c001 ce d0 00    ldx #$d000 ; Load Index X with initial location
0012 c004 a7 00    HERE staa 0,x      ;Store contents of AccA
0013 c006 8b 01          adda #$01 ;Increment A
0014 c008 08          inx          ;Increment Index X
0015 c009 8c d0 0a    cpx #$d00a ;Compare to $d00a
0016 c00c 2d f6          blt HERE  ; If less than, jump to store new value
0017 c00e 7e e0 0a    jmp $e00a ;Jump to BUFFALO
HERE      c004 *0012 0016

```

Program is in C000-C010, therefore the program takes up 17 memory spaces.

```

0001          * IndStor.a11
0002          * J. Chris Perez
0003          * Program uses indexed registering to loop
0004          * through a set of instructions
0005          * to increment value located
0006          * in AccA and store result in consecutive
0007          * Memory locations
0008
0009 c000          org $c000    ;Set location of program in memory
0010 c000 4f          clra          ;Clear Accumulator A
0011 c001 ce d0 00    ldx #$d000 ; Load Index X with initial location
0012 c004 a7 00    HERE staa 0,x      ;Store contents of AccA
0013 c006 8b 01          adda #$01 ;Increment A
0014 c008 08          inx          ;Increment Index X
0015 c009 8c d0 0a    cpx #$d00a ;Compare to $d00a
0016 c00c 2d f6          blt HERE  ; If less than, jump to store new value
0017 c00e 7e e0 0a    jmp $e00a ;Jump to BUFFALO
HERE      c004 *0012 0016

```

This program is in C000-C009, therefore the program takes up 10 memory spaces.

```

0001          * INNOT2.A11 Read word, NOT it, Write it
0002          * Bruce Hoepfner  01 JAN 94
0003          * Chris Perez    14 June 2000
0004          *
0005          * Read an 8-bit word from PortE.
0006          * Complement the word.
0007          * Write the word to PortB.
0008          * Definitions
0009 1004      PORTB  equ $1004
0010 e00a     BUFFALO equ $e00a
0011          *****
0012          * Load program into 8k user RAM
0013 c000      org $C000
0014          * Read 8-bit word from PortE into AccA
0015 c000 b6 10 0a      ldaa $100a
0016          * Complement the word.
0017 c003 43           coma
0018          * Write the word to PortB
0019 c004 b7 10 04     staa PORTB
0020
0021          * Jump back to beginning of program
0022 c007 7e e0 0a     jmp BUFFALO beginning of program
          BUFFALO  e00a *0010 0022
          PORTB    1004 *0009 0019

```

This program takes 13 cycles (6.5×10^{-6} sec) to execute.

| | | | | |
|-----------------------------|----------------------------|----|-------------------------|--|
| | | | 0001 | * INNOT2.A11 Read word, NOT it, Write it |
| | | | 0002 | * Bruce Hoeppe 01 JAN 94 |
| | | | 0003 | * Chris Perez 14 June 2000 |
| | | | 0004 | * |
| | | | 0005 | * Read an 8-bit word from PortE. |
| | | | 0006 | * Complement the word. |
| | | | 0007 | * Write the word to PortB. |
| | | | 0008 | * Definitions |
| From Appendix A: | | | 0009 1004 | PORTB equ \$1004 |
| Instruction addressing mode | Cycles | | 0010 e00a | BUFFALO equ \$e00a |
| | | | 0011 | ***** |
| ldaa | extended | 4 | 0012 | * Load program into 8k user RAM |
| coma | inherent | 2 | 0013 c000 | org \$C000 |
| staa | extended | 4 | 0014 | * Read 8-bit word from PortE into AccA |
| jmp | extended | 3 | 0015 c000 b6 10 0a | ldaa \$100a |
| | | | 0016 | * Complement the word. |
| Total: | | 13 | 0017 c003 43 | coma |
| | | | 0018 | * Write the word to PortB |
| 13 cycles/freq | = 13 / 2MHz | | 0019 c004 b7 10 04 | staa PORTB |
| | = 6.5×10^{-6} sec | | 0020 | |
| | | | 0021 | * Jump back to beginning of program |
| | | | 0022 c007 7e e0 0a | jmp BUFFALO beginning of program |
| | | | BUFFALO e00a *0010 0022 | |
| | | | PORTB 1004 *0009 0019 | |

Laboratory μ p2: Software

Prelab: Create flowcharts and write code for the following:

1. Data Entry – Enter eight 4-bit numbers using a dipswitch (or BCD switch). Read the number when a button is pressed. Store the numbers in sequential memory locations, start at \$D000. Display each number as it is entered.
2. Data Sort —Sort eight 4-bit numbers in memory. (Largest to smallest)
3. Data Sum – Compute the sum of eight 4-bit numbers in memory. Display result in BCD.
4. Combine Routines – Combine the functions of 1,2 and 3 into one program. Use 2 bits of PortA to specify which function to perform. 00=Data Entry, 01=Data Sort, 10=Data Sum, 11=Exit Program

Design Rules:

1. Use proper documentation when creating your source code.

M68HC11EVB I/O limitations: PortB, PortE, PortA (excluding PA7 and PA3)

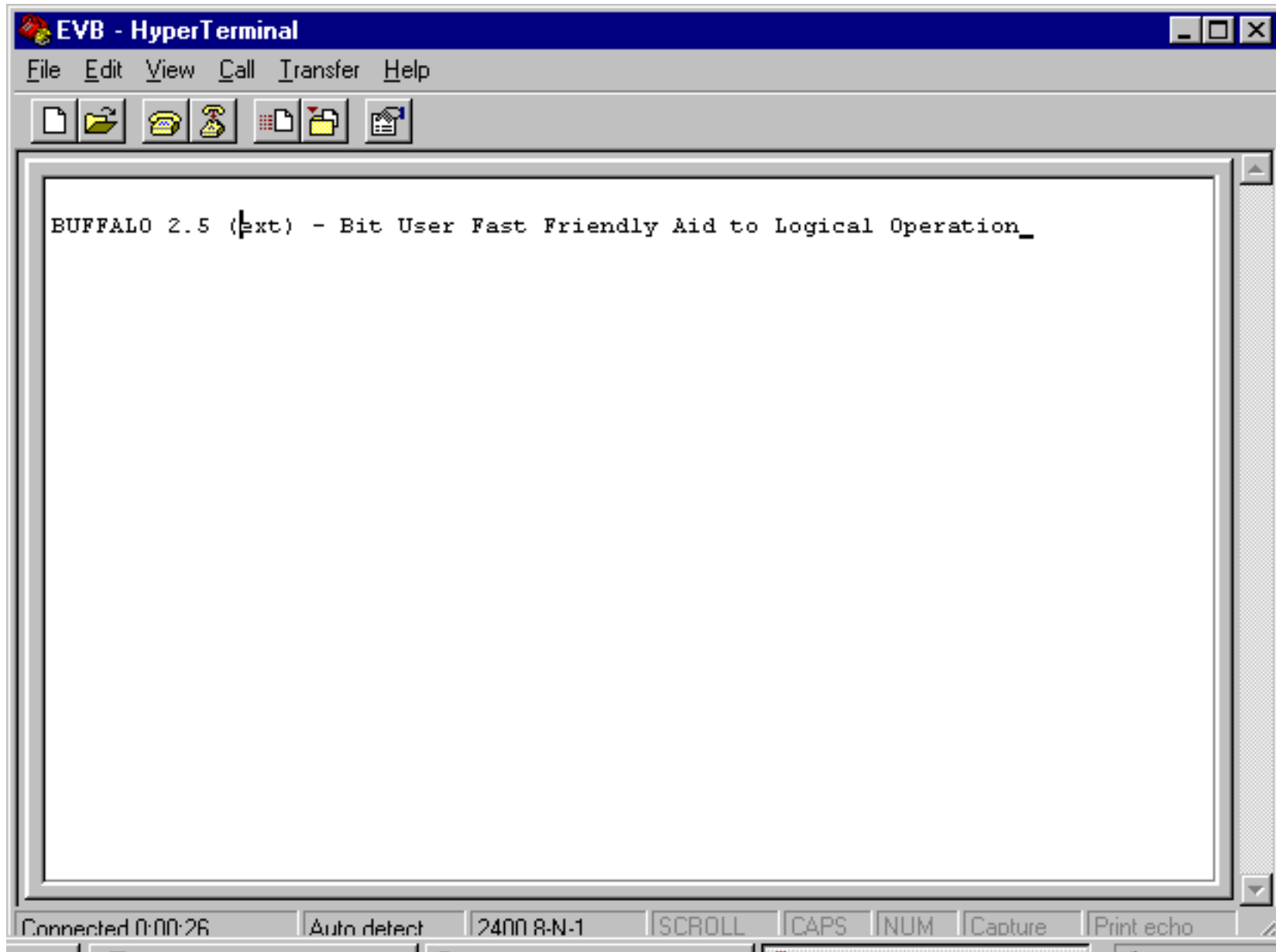
Remember for Prelab: Show schematic diagrams for all external hardware designs.

Open your hyperterminal connection to the EVB.

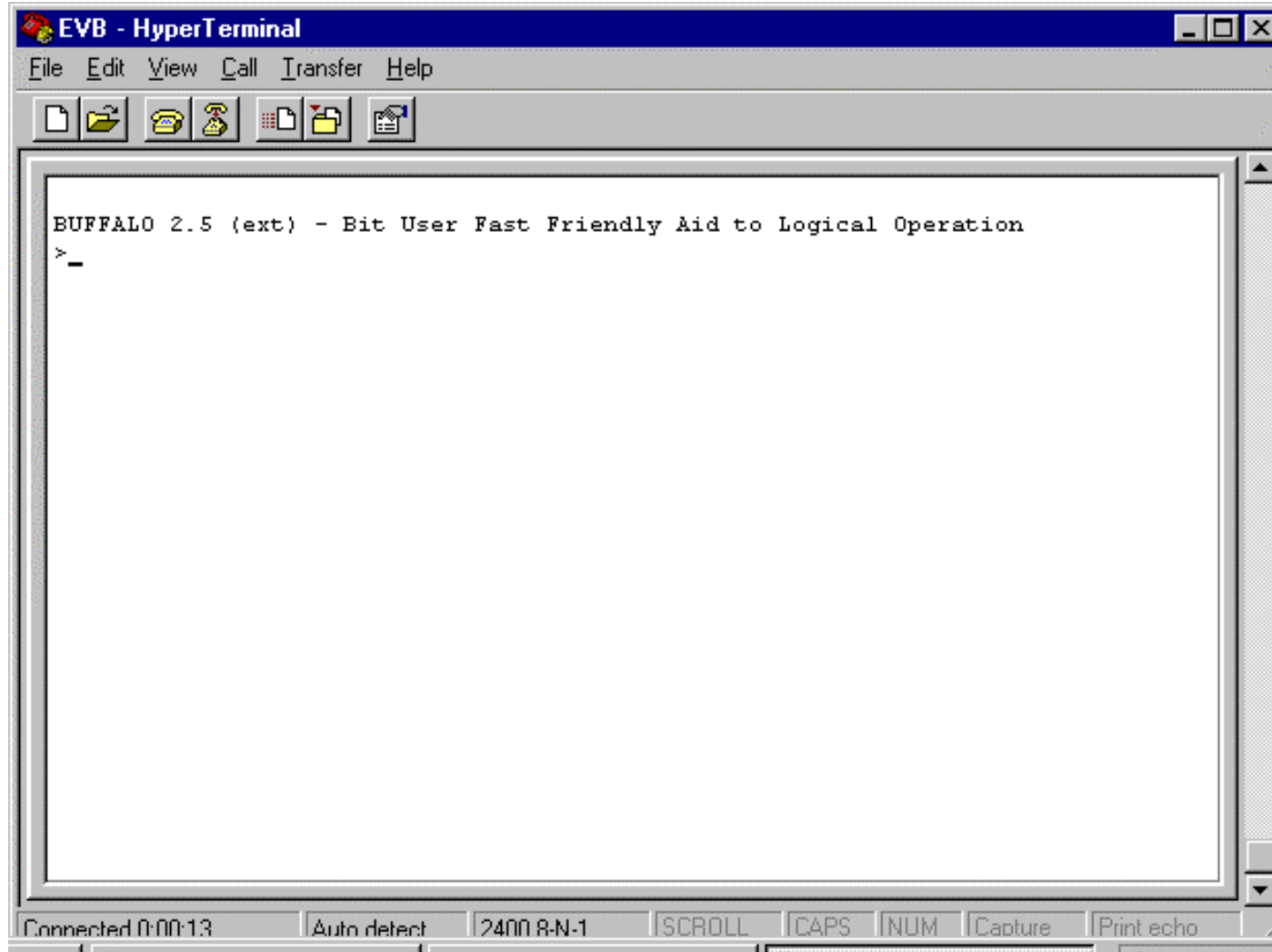
Power up your EVB and press the EVB RESET button.

Note the contents of the screen:

BUFFALO-stands for “Bit User Fast Friendly Aid to Logical Operations”



Press Enter. This brings up the BUFFALO prompt > You are ready to use the 68HC11EVB.



ASM == on-board assembler

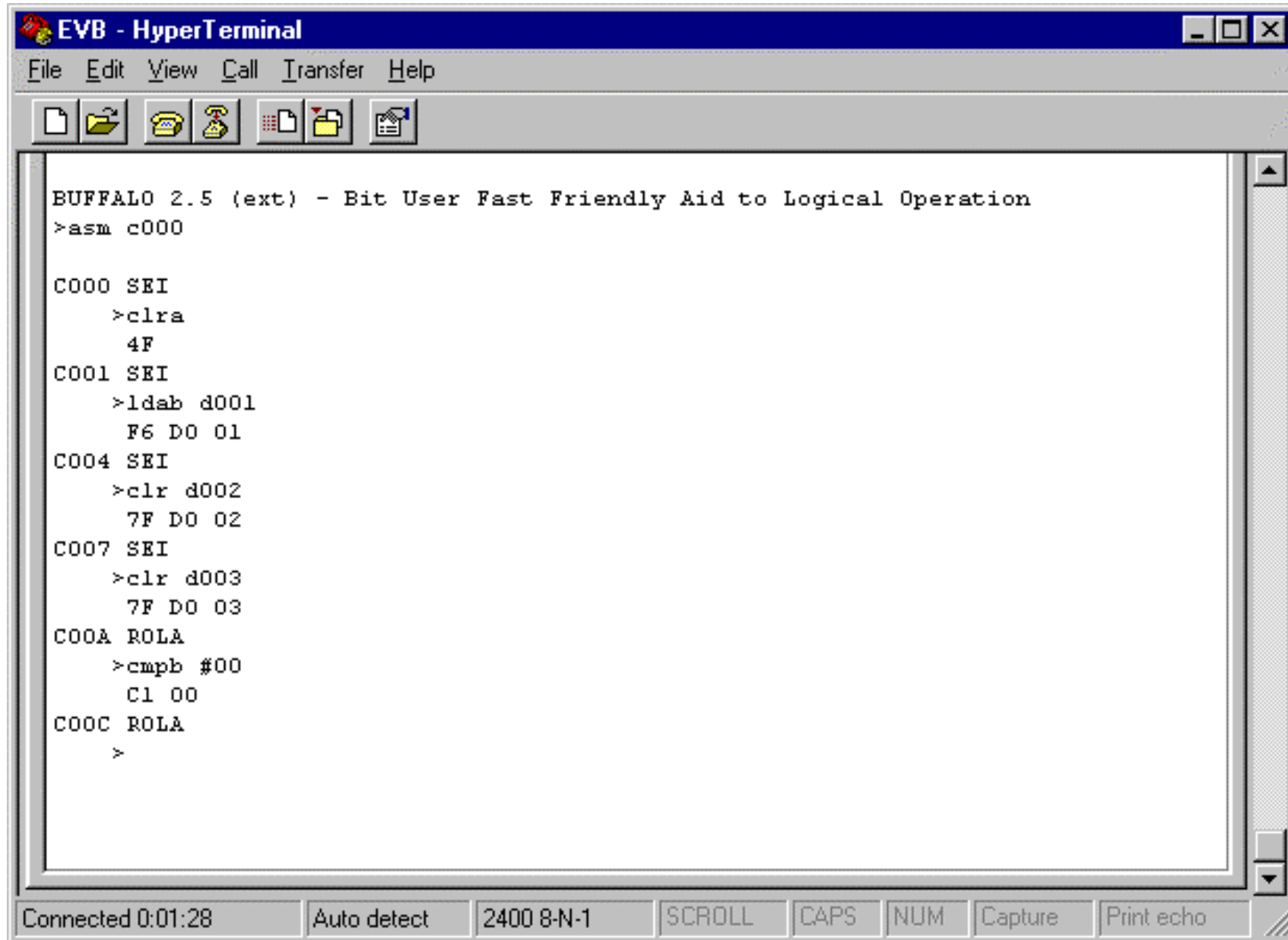
ASM *address*

Displays the assembly language for specified address. User may change the instruction and/or data.

Labels may not be used. Instead, use the actual memory location where the label points to. Use your .LST file for help.

Use <CR> to advance to next line of code.

Use <CTRL-C> to abort on-board assembly.



```
EVB - HyperTerminal
File Edit View Call Transfer Help

BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation
>asm c000

C000 SEI
    >clra
    4F
C001 SEI
    >ldab d001
    F6 D0 01
C004 SEI
    >clr d002
    7F D0 02
C007 SEI
    >clr d003
    7F D0 03
C00A ROLA
    >cmpb #00
    C1 00
C00C ROLA
    >
```

Connected 0:01:28 Auto detect 2400 8-N-1 SCROLL CAPS NUM Capture Print echo

G Go / Execute command

G address

Begins execution of a program at the specific address. The program must provide a jump instruction to get back to the BUFFALO prompt. (jmp \$e00a)

\$E00A is the location of BUFFALO in memory.

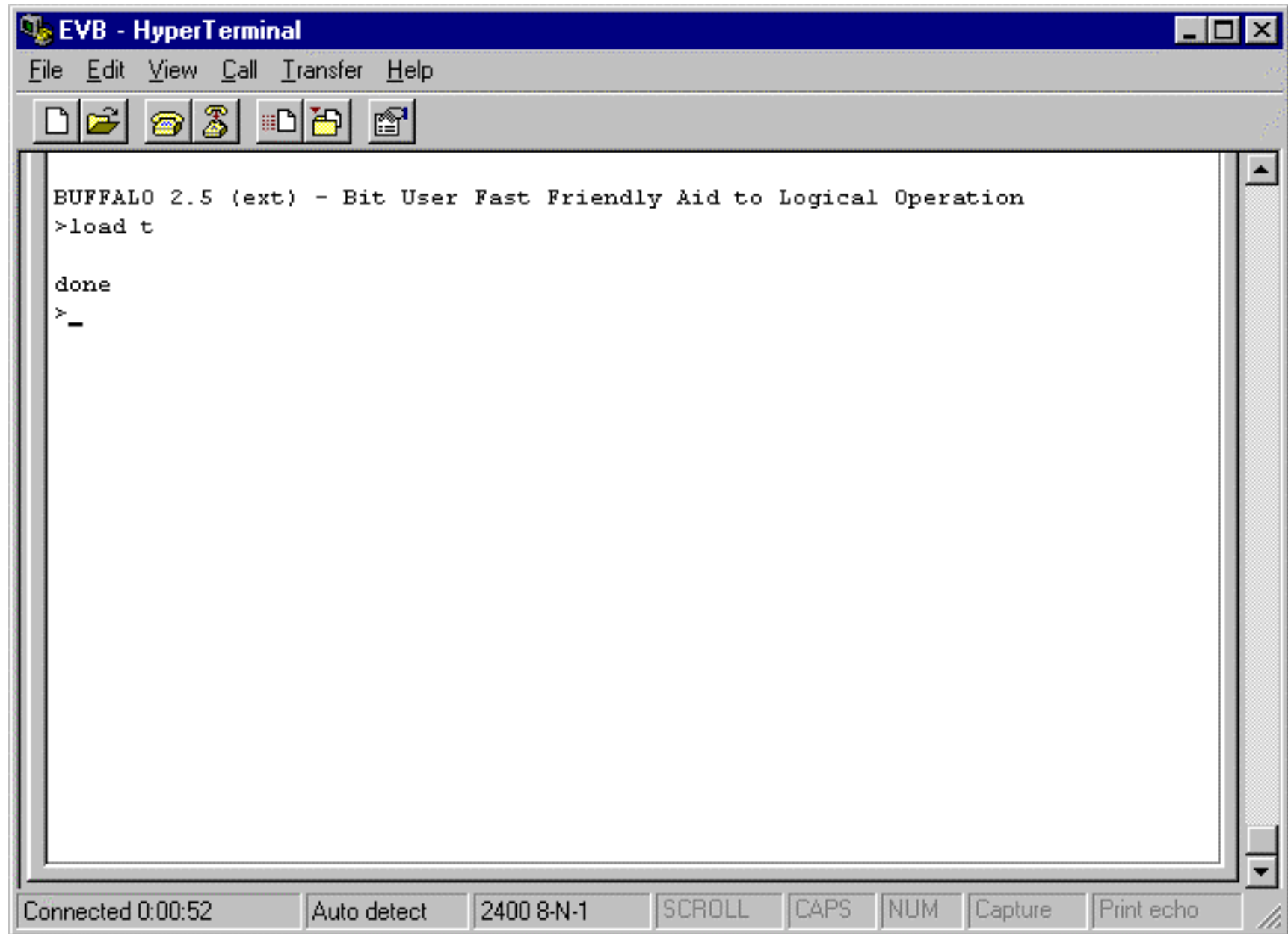
LOAD T

Download an assembled .S19 file from a PC

After entering the LOAD T command, go to the Transfer menu:Send Text File. Choose the .S19 file you wish to load.

When the file is loaded the word done appears on the screen.

You can now verify the file was loaded by using the Memory Display command (MD) and then the G command to run your program.



MD Memory Display: Display memory to terminal screen

MD address_start [address_stop]

Displays 16 bytes per line. Display will begin on an even 16 byte memory boundary. If no address_stop is given, 9 lines will be displayed. If address_stop is less than address_start, one line will be displayed.

```

EVB - HyperTerminal
File Edit View Call Transfer Help

BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation
>load t

done
>md c000

C000 4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0      &
C010 27 F9 8B 01 19 B7 10 04 7E C0 04 B6 B6 B6 B6 B6 '
C020 OF OF OF OF OF OF OF OF 49 49 49 49 49 49 49 49      I I I I I I I I
C030 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C040 OF OF OF OF OF OF OF OF 49 49 49 49 49 49 49 49      I I I I I I I I
C050 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C060 OF OF OF OF OF OF OF OF 49 49 49 49 49 49 49 49      I I I I I I I I
C070 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C080 OF OF OF OF OF OF OF OF 49 49 49 49 49 49 49 49      I I I I I I I I
>

```

Connected 0:04:33 Auto detect 2400 8-N-1 SCROLL CAPS NUM Capture Print echo


```

EVB - HyperTerminal
File Edit View Call Transfer Help

C000 4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0      &
>md c010

C010 27 F9 8B 01 19 B7 10 04 7E C0 04 B6 B6 B6 B6 B6 '
C020 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49 49      IIIIIIII
C030 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C040 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49 49      IIIIIIII
C050 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C060 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49 49      IIIIIIII
C070 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
C080 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49 49      IIIIIIII
C090 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6 B6 B6
>md c000 0

C000 4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0      &
>md c000 c00f

C000 4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0      &
>md c000 c010

C000 4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0      &
C010 27 F9 8B 01 19 B7 10 04 7E C0 04 B6 B6 B6 B6 B6 '
>

```

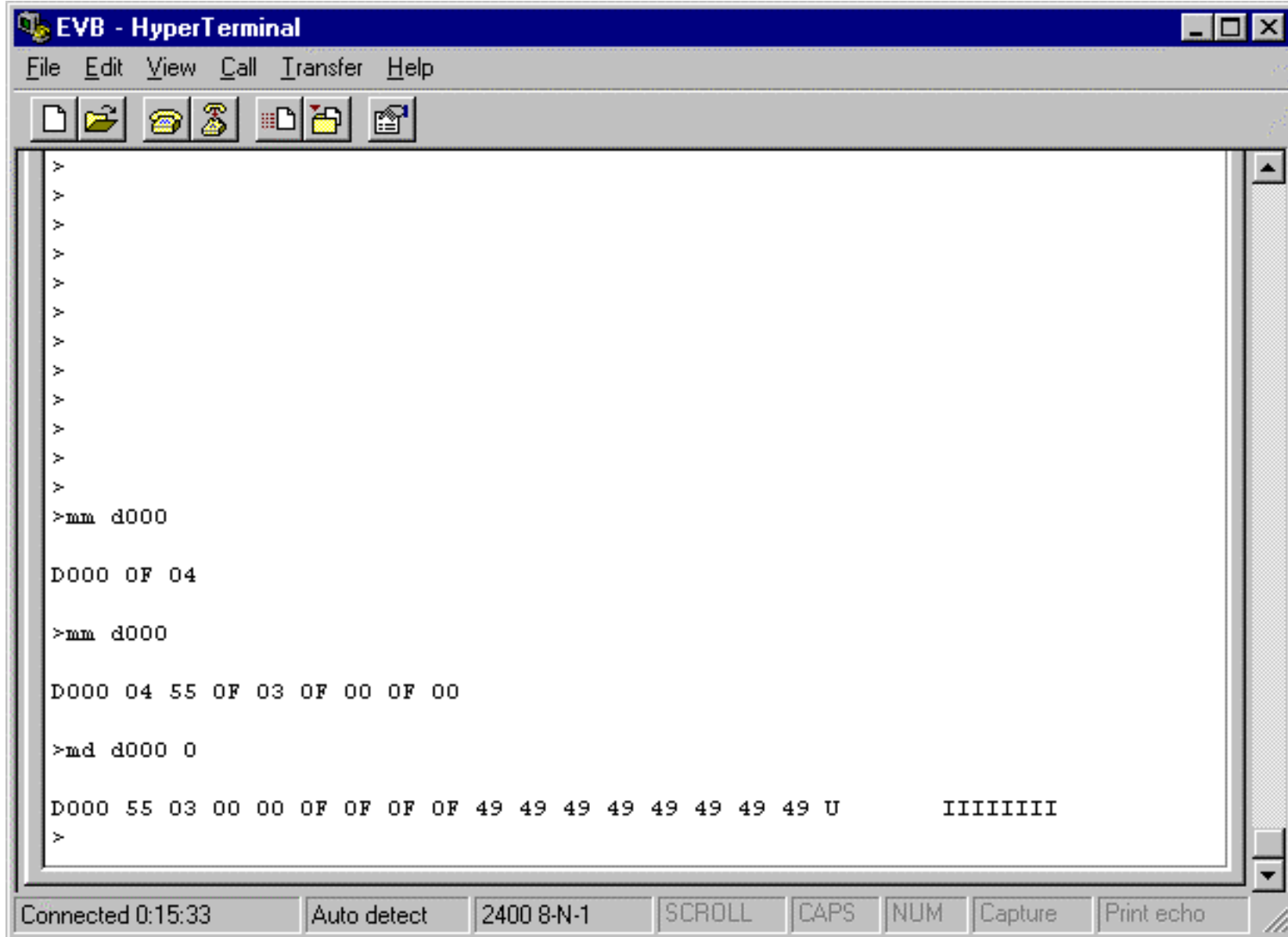
Connected 0:10:27 Auto detect 2400 8-N-1 SCROLL CAPS NUM Capture Print echo

MM Memory modify: Display and modify memory contents.

MM *address*

Displays memory and gives the user a chance to modify it.

Use the <SPACE> key to advance to the next byte. Use CTRL-H to backup one byte. Use <CR> to return to the BUFFALO prompt.



RM Register Modify: Display and modify 68HC11 registers.

RM [*p,y,x,a,b,c,s*]

Displays the contents of the 68HC11's registers. Also gives the user the chance to modify them.

Registers include:

P – program counter

Y – index register Y

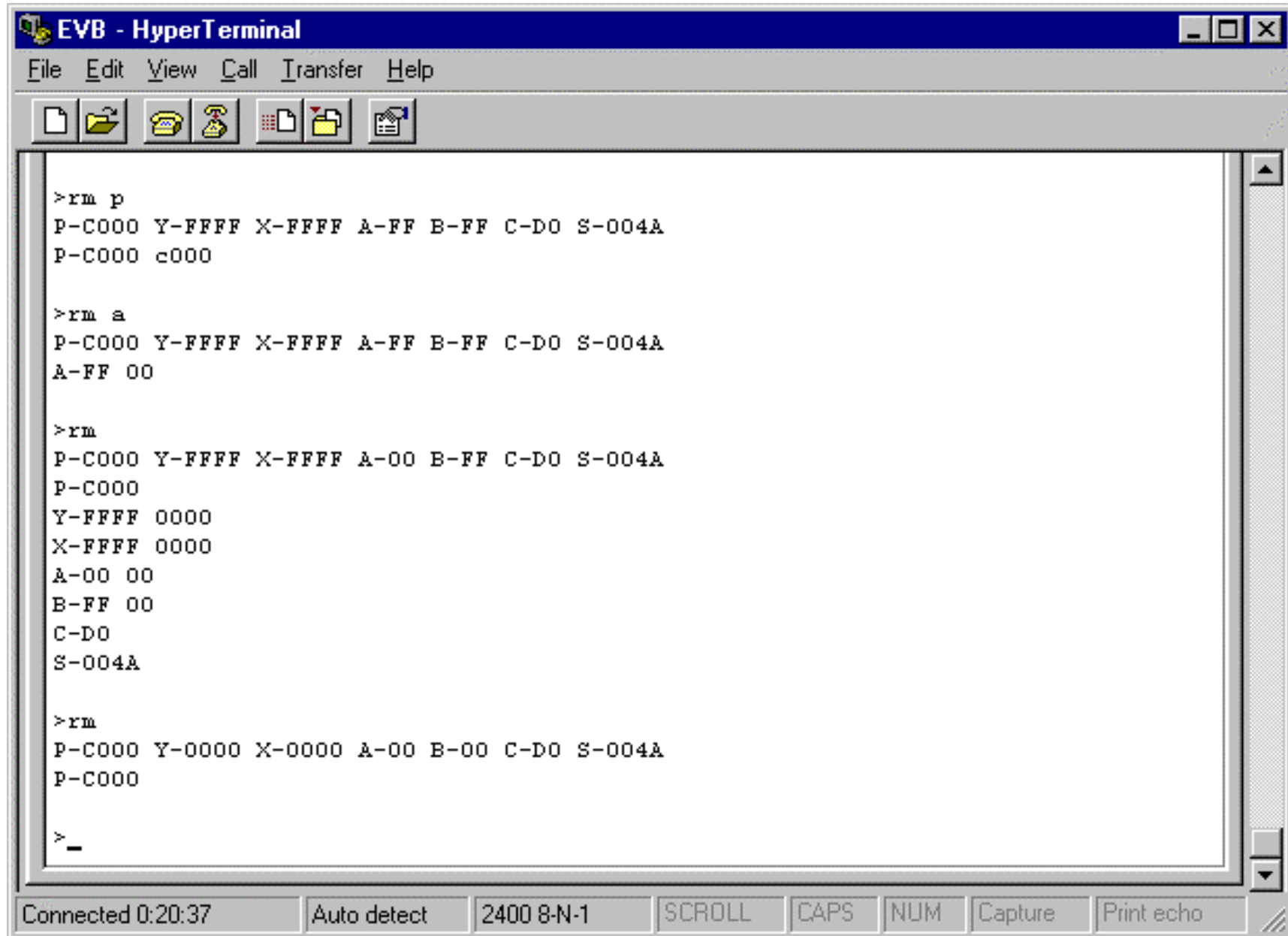
X – index register X

A – Accumulator A

B – Accumulator B

C – Condition code register

S –Stack pointer



The screenshot shows a HyperTerminal window titled "EVB - HyperTerminal". The window has a menu bar with "File", "Edit", "View", "Call", "Transfer", and "Help". Below the menu bar is a toolbar with icons for file operations. The main text area contains the following text:

```
>rm p
P-C000 Y-FFFF X-FFFF A-FF B-FF C-D0 S-004A
P-C000 c000

>rm a
P-C000 Y-FFFF X-FFFF A-FF B-FF C-D0 S-004A
A-FF 00

>rm
P-C000 Y-FFFF X-FFFF A-00 B-FF C-D0 S-004A
P-C000
Y-FFFF 0000
X-FFFF 0000
A-00 00
B-FF 00
C-D0
S-004A

>rm
P-C000 Y-0000 X-0000 A-00 B-00 C-D0 S-004A
P-C000

>_
```

At the bottom of the window, there is a status bar with the following information: "Connected 0:20:37", "Auto detect", "2400 8-N-1", "SCROLL", "CAPS", "NUM", "Capture", and "Print echo".

T Trace Instructions

T [n]

The trace instruction allows the user to execute a program n instructions at a time. The user must set the program counter to the correct starting address before using the trace command. The machine code for the instruction will be displayed along with the registers after each instructions.

```
EVB - HyperTerminal
File Edit View Call Transfer Help

>v
>v
>v
>t 1

Op- 4F
P-C001 Y-0000 X-0000 A-00 B-00 C-C4 S-004A
>

Op- B7
P-C004 Y-0000 X-0000 A-00 B-00 C-C4 S-004A
>

Op- F6
P-C007 Y-0000 X-0000 A-00 B-00 C-C4 S-004A
>

Op- C4
P-C009 Y-0000 X-0000 A-00 B-00 C-C4 S-004A
>
_

Connected 0:23:36 Auto detect 2400 8-N-1 SCROLL CAPS NUM Capture Print echo
```