EECE416 :Microcomputer Fundamentals and Design

68000 Programming Techniques

with EASY68K

Programming Techniques

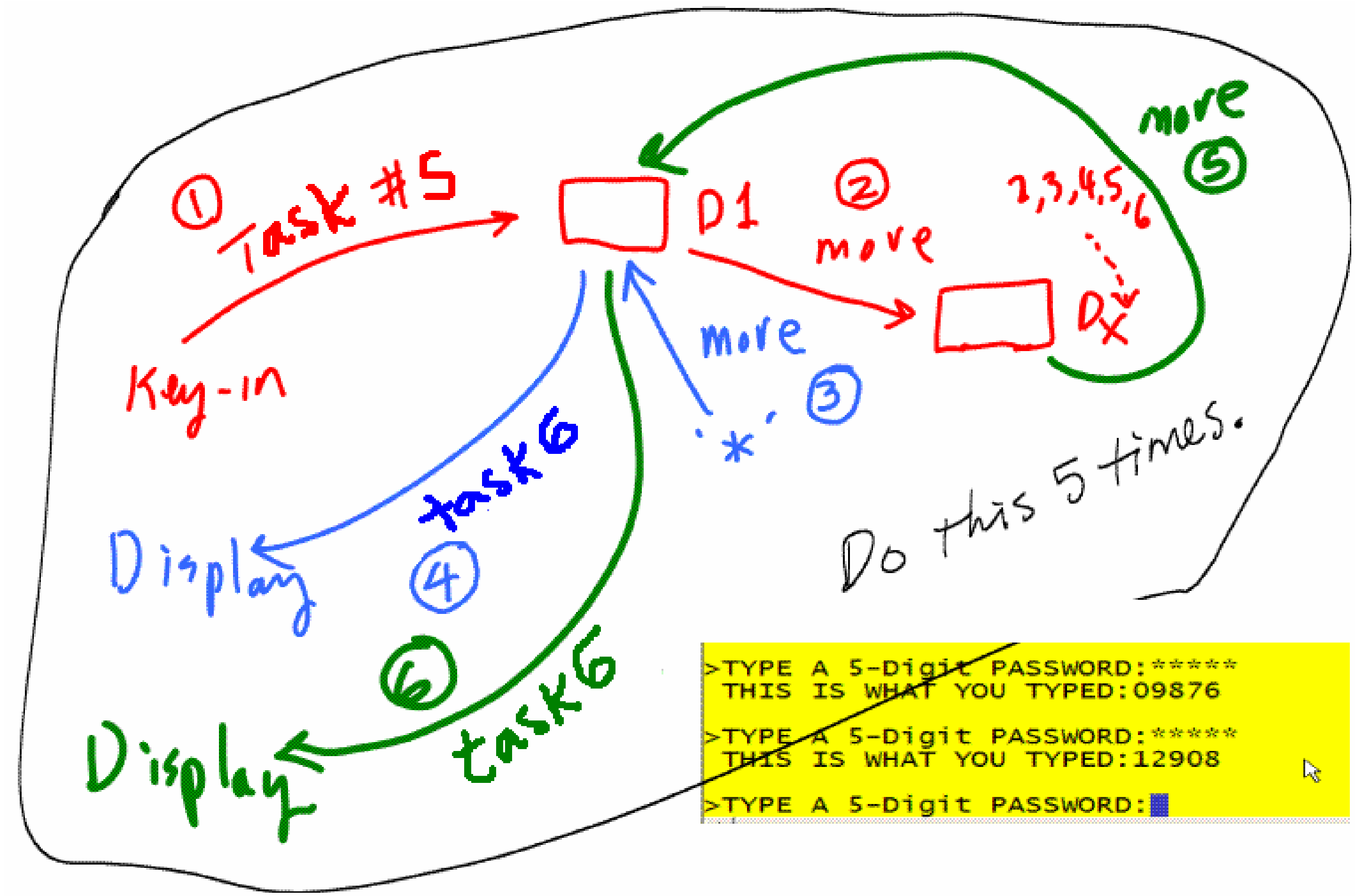
- Subroutines and Parameter Passing
- Data gathering
- Searching Data Table
- String Operations
- Sorting
- Computational Routines
- Number Conversion
- Examples

Exercise I

- TrapExample.x68
  - Using different TRAPs for Key-In and Display
    - Trap task 5
      - Read a character from keyboard
      - Stored the keyed-in in the D1.B
    - Trap task 6
      - Display a character stored in D1.B
    - Trap task 0 (with CR.LF)/1 (w/o CR.LF)
      - Display a string of characters whose starting address is stored in A1 register
      - Display of the string continues until it meets number 0 [zero]
    - Trap task12
      - Key echo-off (with D1.B=0)
      - Key echo-on (with D1.B=non zero value)
  - A character guessing game

Exercises (Password Echo)

- P-ECHO.x68
  - Accept 5-digit number, and display * for each digit
  - And display at the next line the password



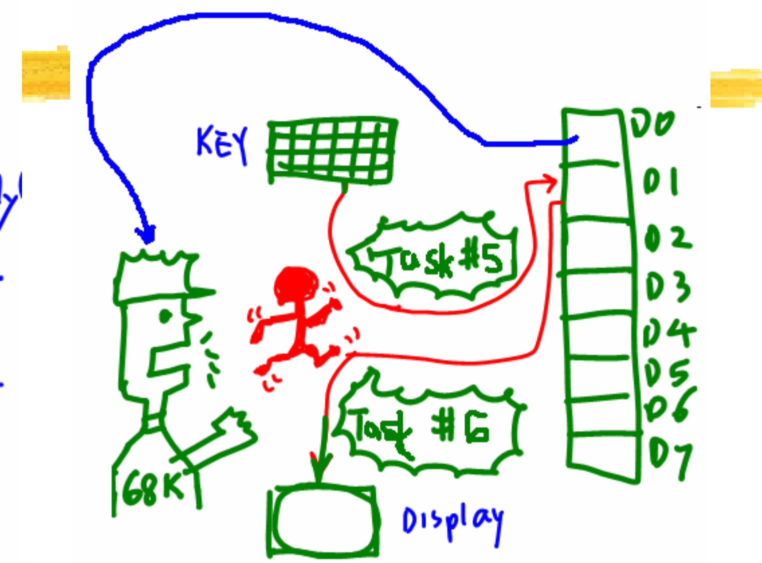
Tell me why we have to move around.

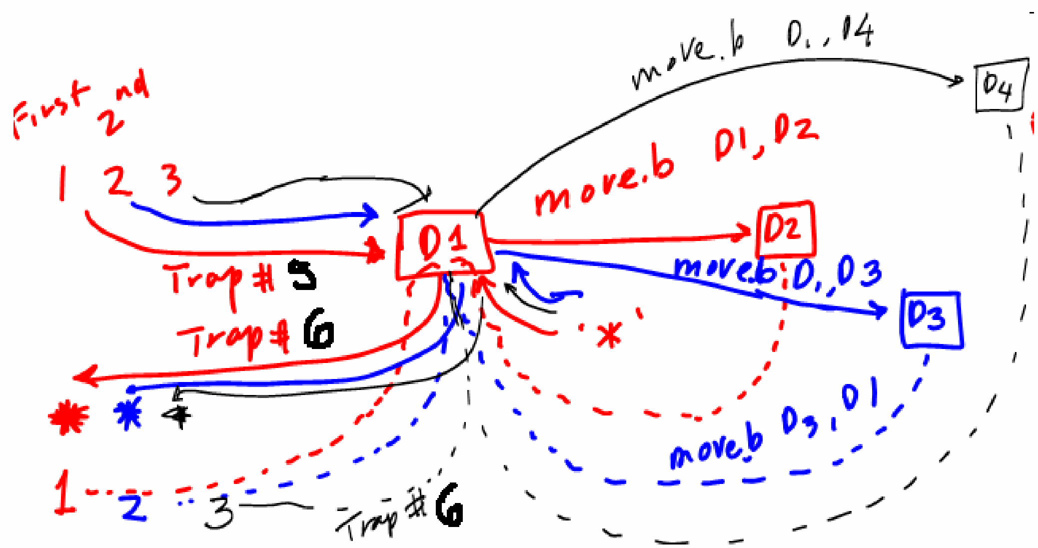
① Only D_1 is used as a receptionist:
Input and Output should pass through the receptionist.

② The Receptionist (D_1) handlles only 1 visitor (1 character). If you input 2 characters, only the latest one is registered and stored in D_1

③ Fortunately, there are 7 additional Rooms to Keep the ristors: $D_0, D_2, D_3, D_4, D_5, D_6, D_7$.

④ So, for multiple characters, they can be stayed (after registering with D1, receptionist) at one of the rooms.

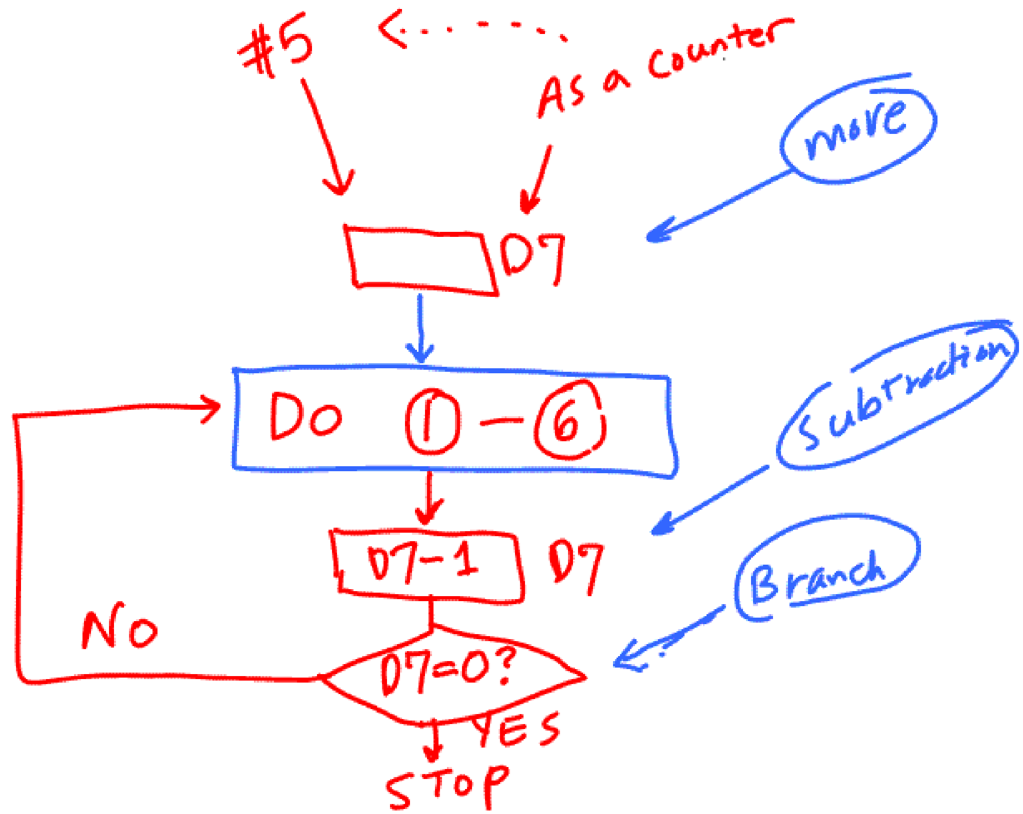




Exercises - continued

- Revision of P-ECHO.asm to C-ECHO.asm
  - Allow only 5 times of Password Tries for ATM access

```
++ ATM ACCESS SCREEN ++
++ WARNING: MAX NUMBER OF TRIES IS 5 ++

>TYPE A 5-Digit PASSWORD:*****
 THIS IS WHAT YOU TYPED:12342

>TYPE A 5-Digit PASSWORD:*****
 THIS IS WHAT YOU TYPED:12312

>TYPE A 5-Digit PASSWORD:*****
 THIS IS WHAT YOU TYPED:31231

>TYPE A 5-Digit PASSWORD:*****
 THIS IS WHAT YOU TYPED:31231

>TYPE A 5-Digit PASSWORD:*****
 THIS IS WHAT YOU TYPED:12312

ATM MESSAGE: YOUR CARD IS CONFISTICATED
```



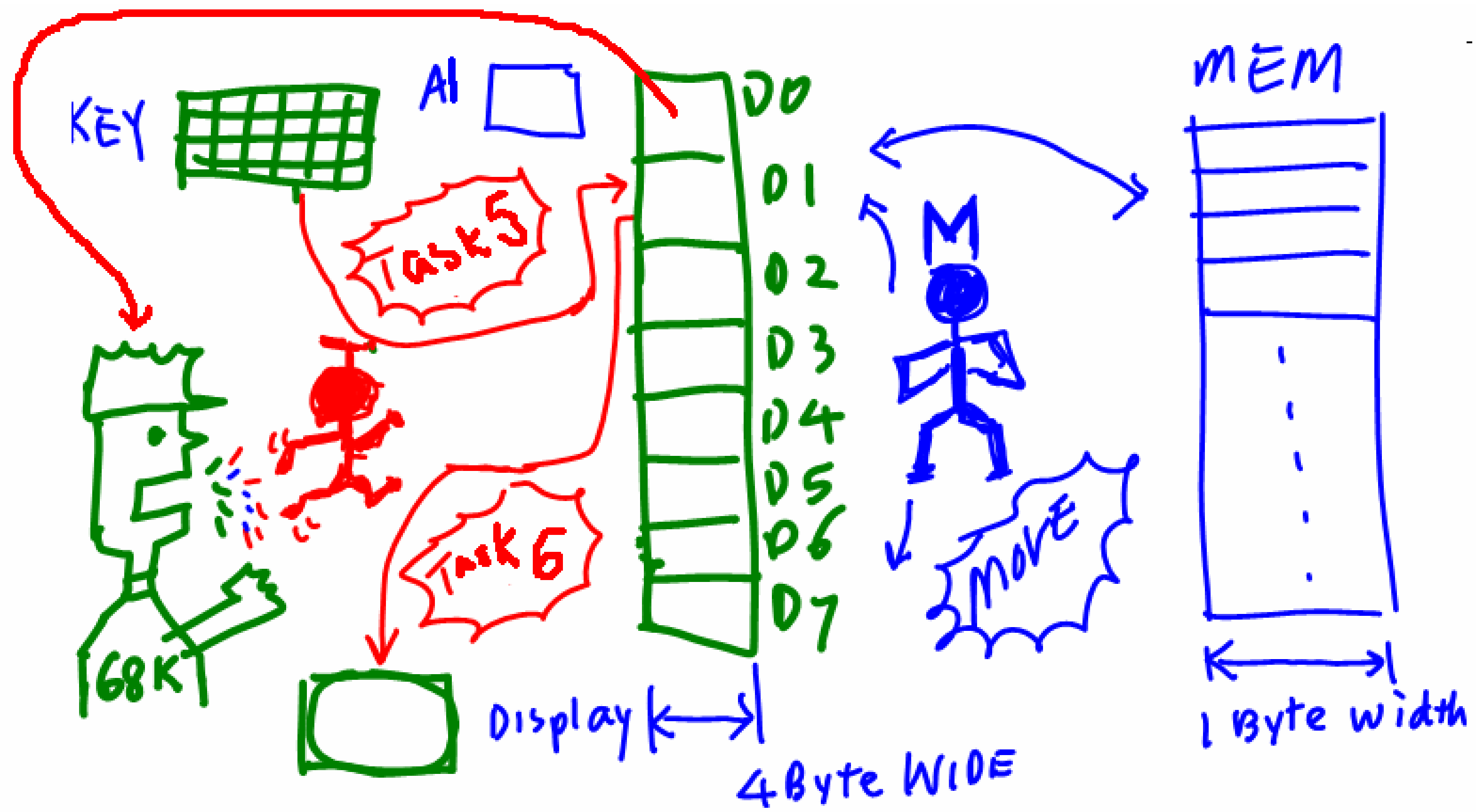
3 Subroutines for TRAP business

- RCHR
  - Read a character
  - Input: Key-in
  - Output: Key-in is stored in D1
- PCHR
  - Print a character
  - Input: a character stored in D1
  - Output: Display on Monitor
- EOFF
  - Echo-Off Declaration
  - Called once at top
  - Input: None
  - Output:None

```
; SUBROUTINES
RCHR    MOVE.B #5,  D0
        TRAP   #15
        RTS

PCHR    MOVE.B #6,  D0
        TRAP   #15
        RTS

EOFF    MOVE.B #0,D1
        MOVE.B #12,D0
        TRAP   #15
        RTS
```

Moving between Data registers and Memory

- What if you need more than 7 long rooms?
  - **Well, in memory, there are many byte rooms. Millions!**

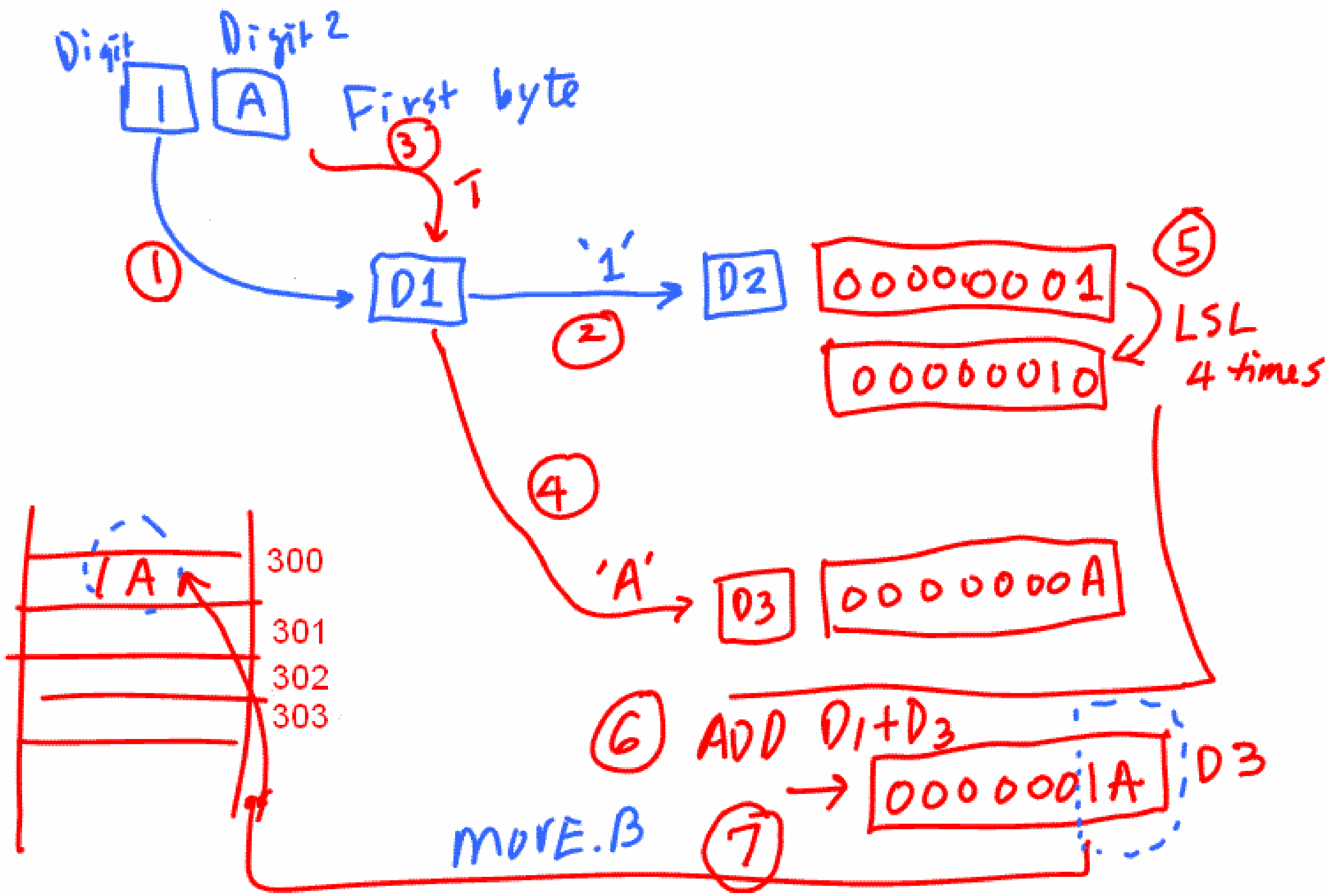


Exercise II (Moving to/from MEM)

- Storing Data Into Memory

```
THIS PROGRAM STORES 4-Byte HEX NUMBER to ADDRESS $300
From High to Low (Use Capital for Letter Digits)
TYPE A 2-Digit HEx Byte
12

TYPE A 2-Digit HEx Byte
8F

TYPE A 2-Digit HEx Byte
9A

TYPE A 2-Digit HEx Byte
6C
```

Is this what will happen?



Exercise II-continued

- This is what we want.

```
Type a Hex Byte: 12
Type a Hex Byte: 3F
Type a Hex Byte: 3A
Type a Hex Byte: C3
```

300	12
301	3F
302	3A
303	C3
304	

- This is what we will have. Why?

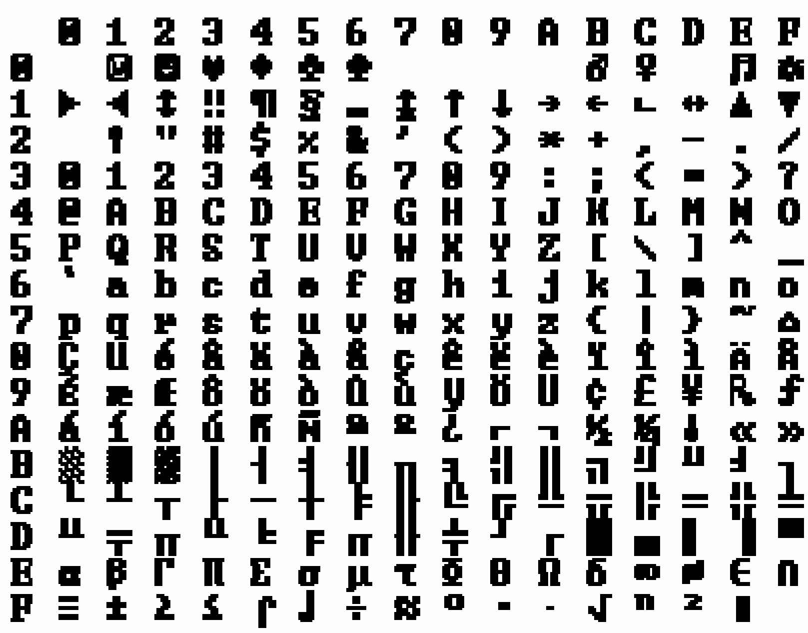
```
Type a Hex Byte: 12
Type a Hex Byte: 3F
Type a Hex Byte: 3A
Type a Hex Byte: C3
```

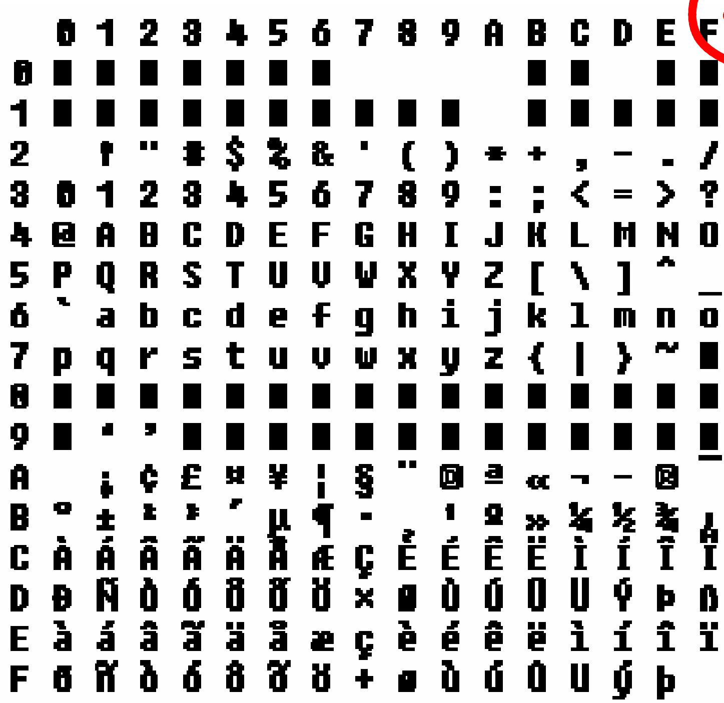
300	42
301	76
302	71
303	63
304	

31	00 00 00 31	00 00 03 10
32	00 00 00 32	00 00 00 32
		42

ASCII Code Chart

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	





ASCII-to-HEX Conversion

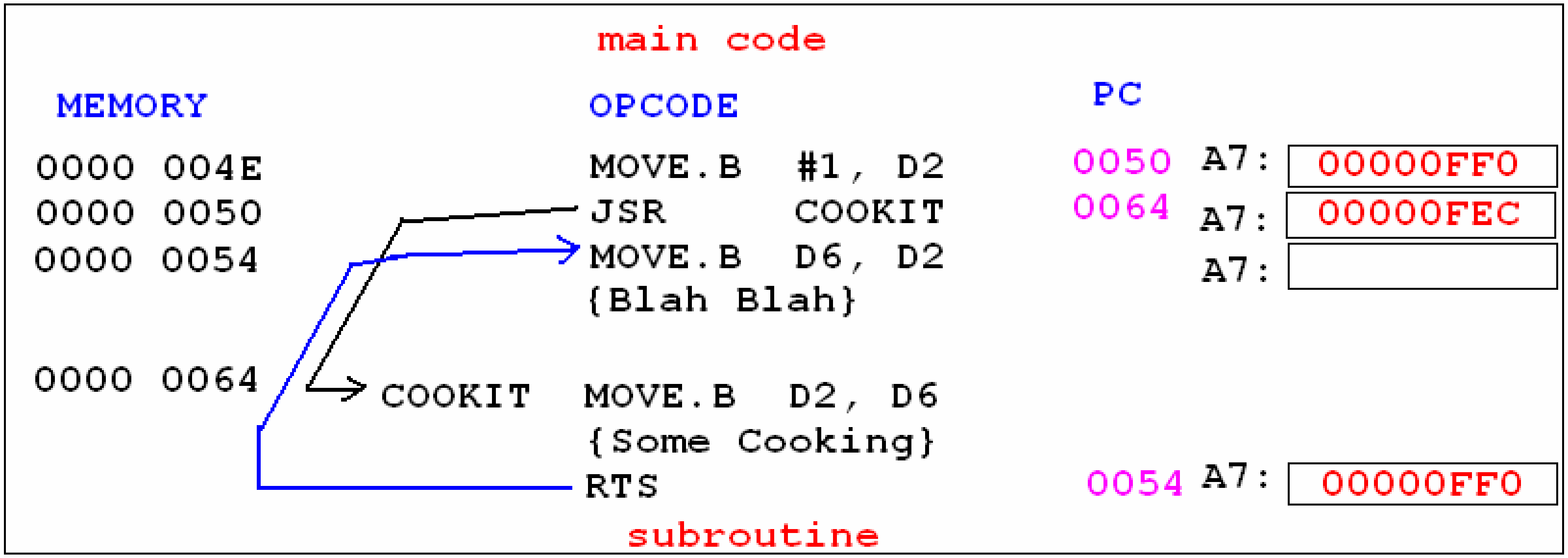
*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

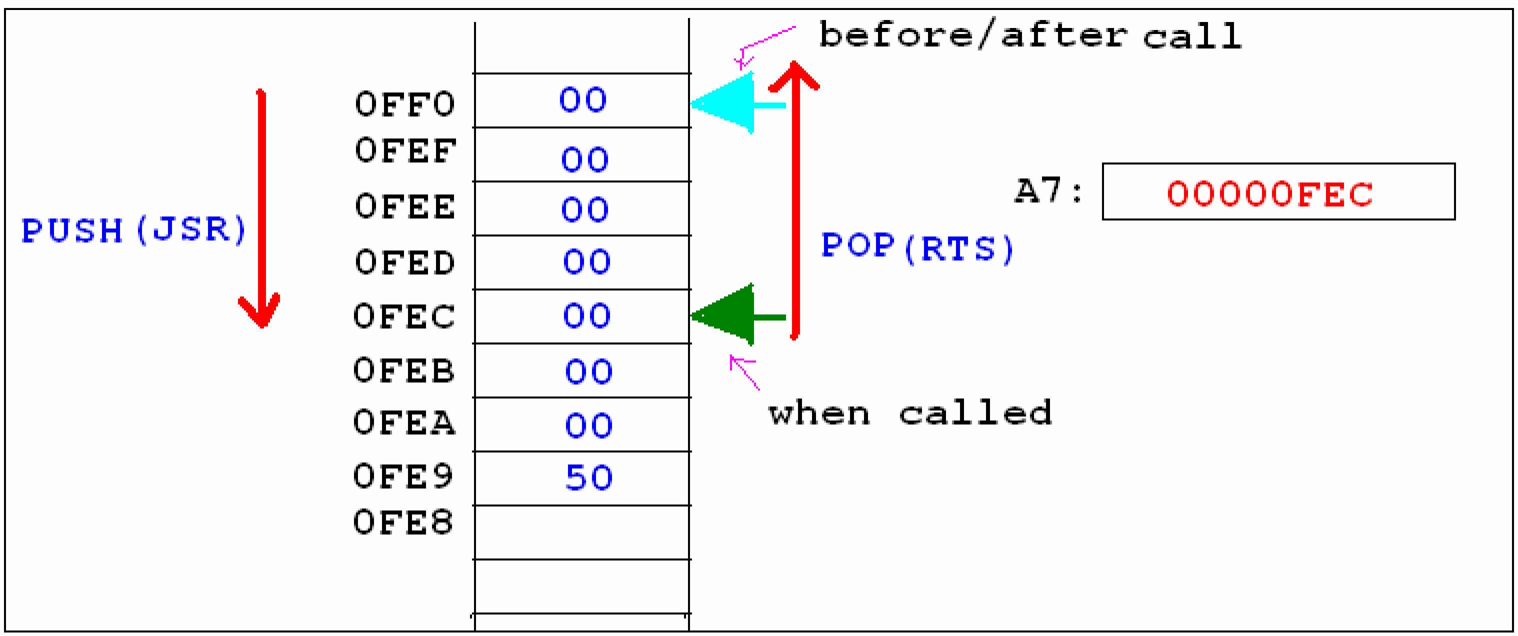
- Task #5
  - A Character (in ASCII code (of a Byte size)) in D1
- Conversion of the Byte in ASCII into a Hex Number
- Numeric or Alpha?
  - $30 - $39 → D1=D1-$30
  - $41 - $46 → D1=D1-$37
  - All others → “error message”

Subroutine and Stack

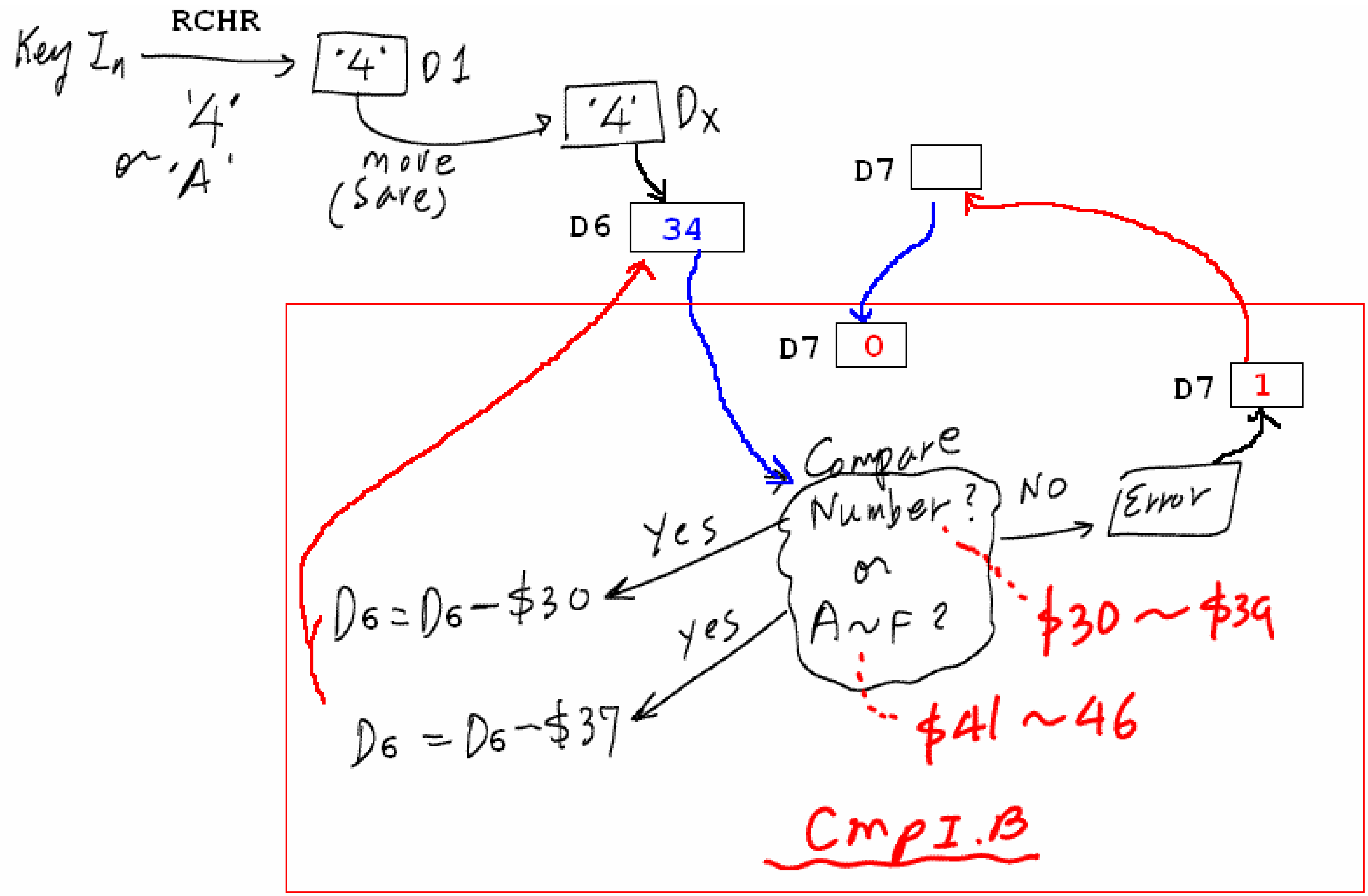
- Subroutine
  - Name= label
  - Ends with **RTS**
- Calling
  - Call by **JSR** or **BSR**
  - Changing PC to the Label (or starting address) of a Subroutine
  - Program should know the return address after visiting the subroutine
- Stack
  - The return address (the address just after the calling instruction) is stored in the Stack
  - Stack is also in the **Memory** (size of Long Word) **starting @00000FF0 and decreasing. So, program code should not mess with stack memory area.**
  - The Stack address is stored in Address register, A7 ("stack pointer")
  - LIFO (Last In First Out) Structure
  - PUSH and POP
- PC gets "Address for next instruction" at the memory location pointed by A7

Subroutine and Stack





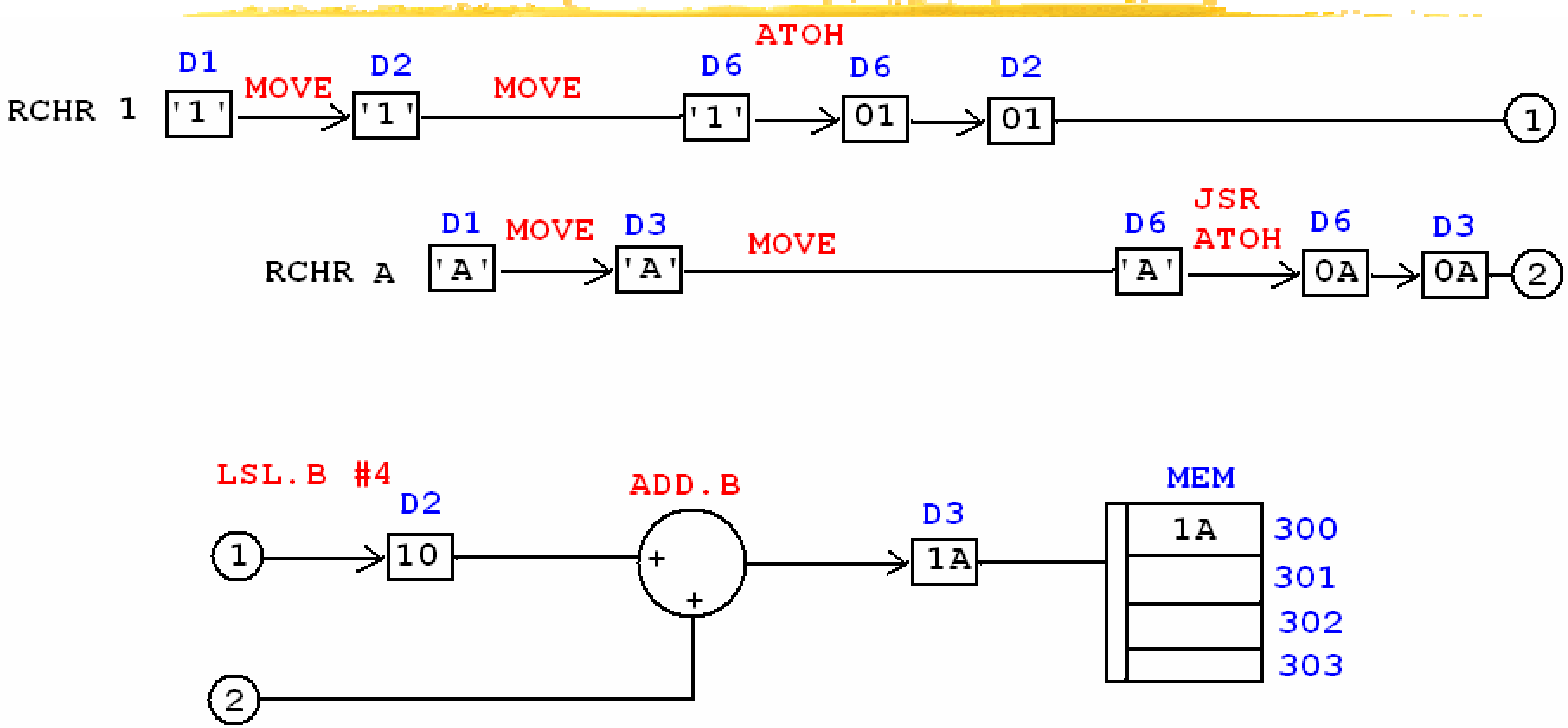
ATOH: ASCII to HEX Conversion Subroutine



Subroutine ATOH (ASCII-to-Hex)

```
;Subroutine ASCII to HEX ================================
;parameter is transferred to D6
AtoH     MOVE.B   #0,D7                    Flagging for non-hex character encounter
         CMPI.B   #$30, D6                 ;Numeric or Alpha
         BLT.B    ERR
         CMP.B    #$39, D6                 ;$30 - $39 for number
         BGT.B    ALPHA
         SUBI.B   #$30, D6
         RTS

ERR      MOVE.B   #80,D1
         MOVE.B   #1,D0
         MOVEA.L  #ERROR, A1
         TRAP     #15
         MOVE.B   #1,D7                    If Error, read next byte
         RTS

ALPHA    CMPI.B   #$41, D6
         BLT.B    ERR
         CMPI.B   #$46, D6
         BGT.B    ERR                      ;$41 - $46 for [A-F]
         SUBI.B   #$37, D6
         RTS
```

Code structure for HEX-to-MEM



Running Result

Sim68K I/O

```
THIS PROGRAM STORES 4-Byte HEX NUMBER to ADDRESS $300
From High to Low (Use Capital for Letter Digits)
TYPE A 2-Digit HEX Byte: 12
TYPE A 2-Digit HEX Byte: 5A
TYPE A 2-Digit HEX Byte: E9
TYPE A 2-Digit HEX Byte: 3B
END
```

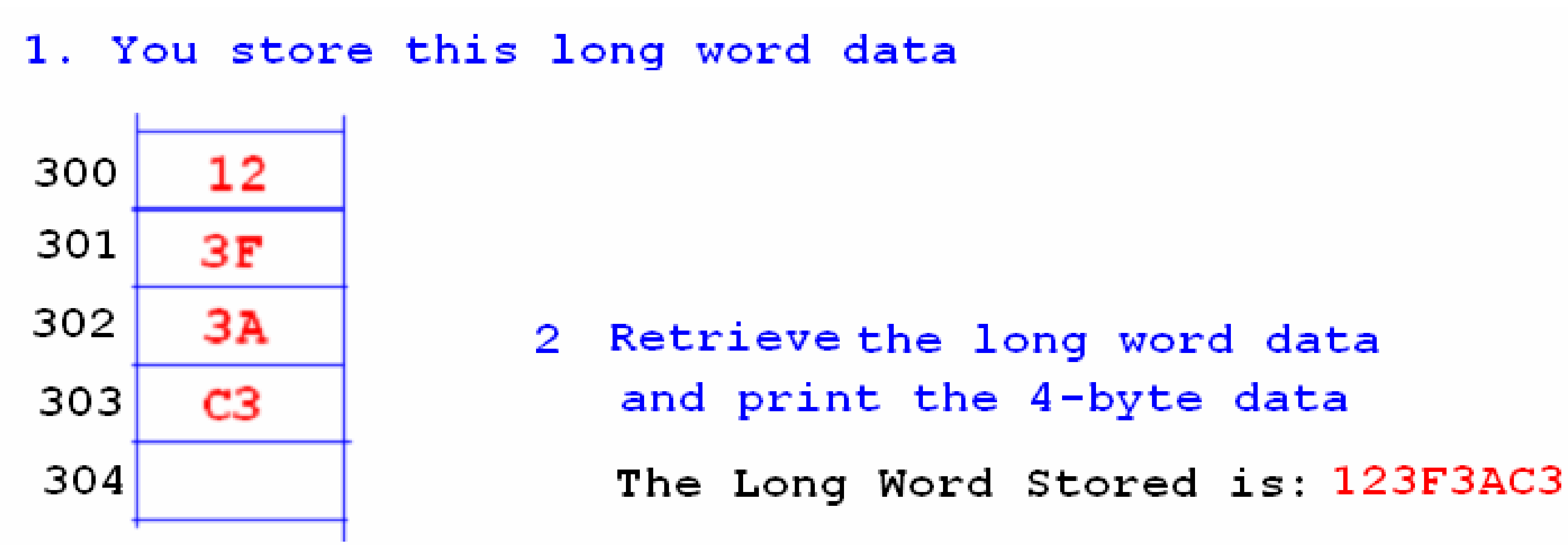
68000 Memory

```
Address:        From: 00000000   To: 00000000   Bytes: 00000000   Copy   Fill   Save
00000210   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00000210: 54 4F 52 45 53 20 34 2D 42 79 74 65 20 48 45 58 TORES 4-Byte HEX
00000220: 20 4E 55 4D 42 45 52 20 74 6F 20 41 44 44 52 45  NUMBER to ADDRE
00000230: 53 53 20 24 33 30 30 00 0D 0A 54 59 50 45 20 41 SS $300---TYPE A
00000240: 20 32 2D 44 69 67 69 74 20 48 45 58 20 42 79 74  2-Digit HEX Byt
00000250: 65 3A 20 00 0D 0A 46 72 6F 6D 20 48 69 67 68 20 e: ---From High
00000260: 74 6F 20 4C 6F 77 20 28 55 73 65 20 43 61 70 69 to Low (Use Capi
00000270: 74 61 6C 20 66 6F 72 20 4C 65 74 74 65 72 20 44 tal for Letter D
00000280: 69 67 69 74 73 29 00 0D 0A 55 6E 73 70 65 63 69 igits)---Unspeci
00000290: 66 69 65 64 20 43 68 61 72 61 63 74 65 72 20 45 fied Character E
000002A0: 6E 63 6F 75 6E 74 65 72 64 00 0D 0A 45 4E 44 00 ncounterd---END-
000002B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000002C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000002D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000002E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000002F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000300: 12 5A E9 3B FF FF FF FF FF FF FF FF FF FF FF FF -Z-;------------
00000310: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000320: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000330: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000340: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000350: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000360: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000370: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
```

Row
Page
Live

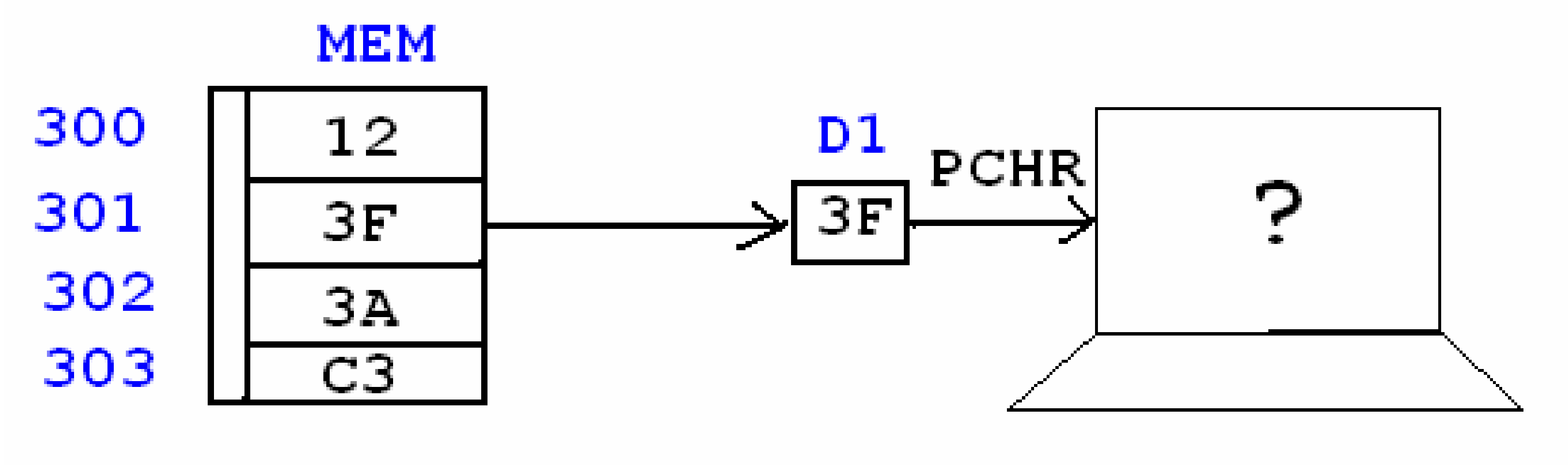
Exercise III

- Problem: Retrieve the long word (I.e., 4 bytes) stored at the location starting at $300, and print each byte, from highest to lowest, on the computer screen.
- This is what we want.



- Need: Conversion of HEX to ASCII (HtoA)

Naïve Approach (without HtoA)

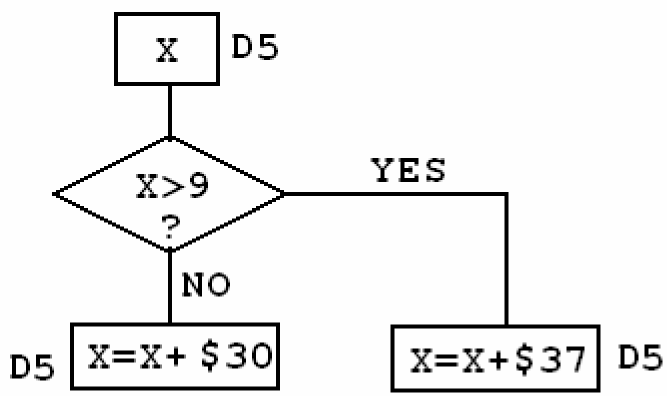


- ASCII Treatment
  - To Monitor
  - From Keyboard
  - Through D1
  - PCHR & RCHR

Hex to ASCII Conversion (HtoA)

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

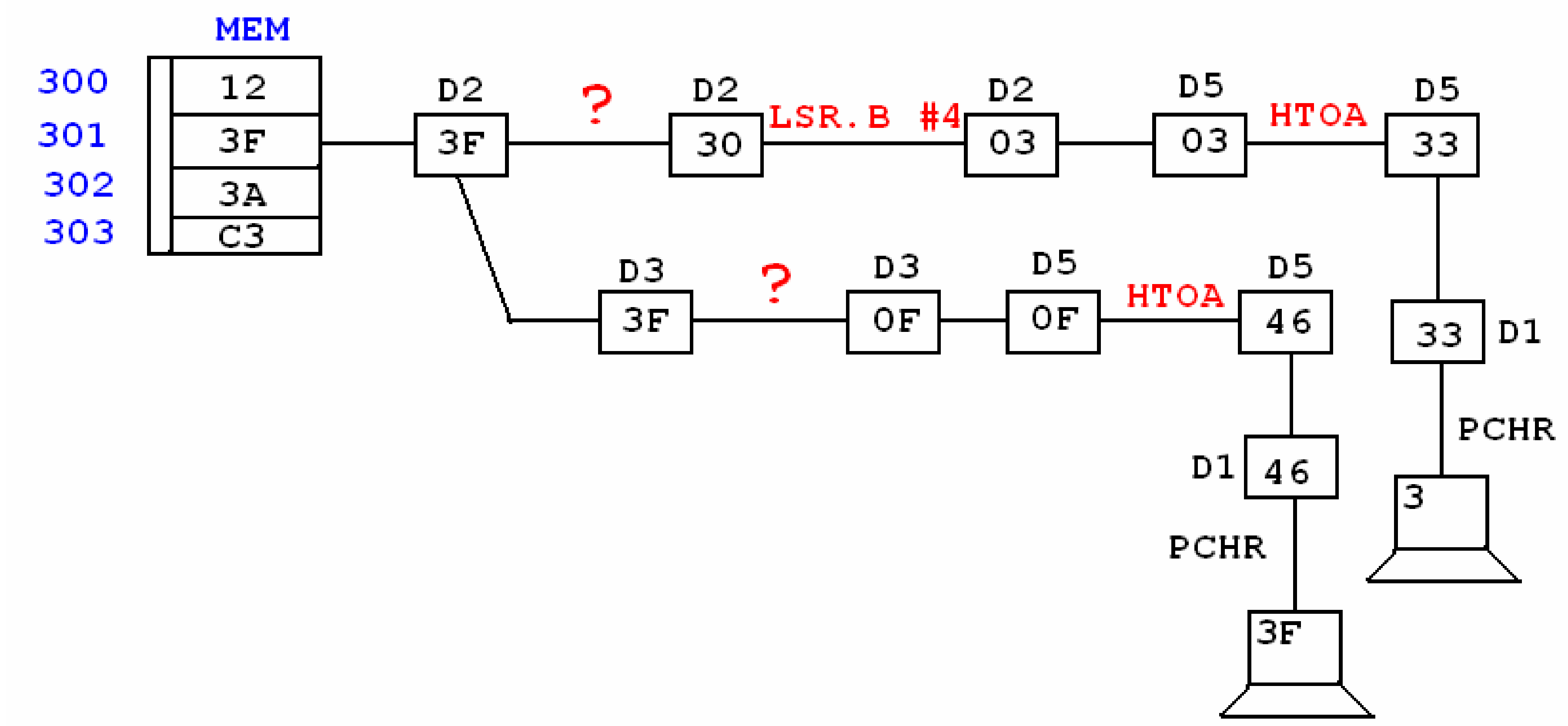
```
HEX     ASCII
---     -----
0        $30  }
1        $31  }   ASC=HEX + $30
9        $39  }
A        $41  }
B        $42  }   ASC=HEX + $37
```



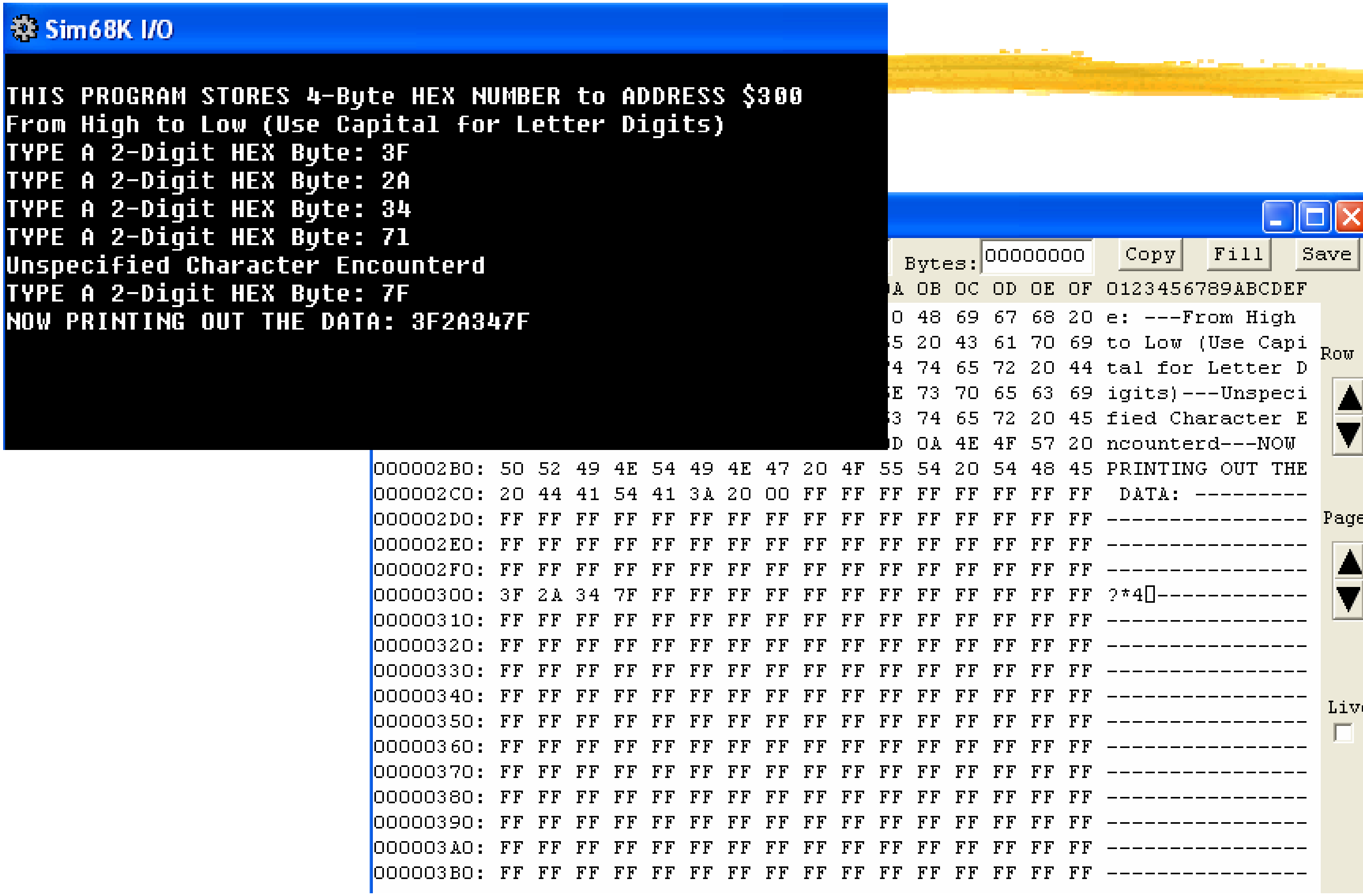

```
;== HtoA ===============================
;D5 is the parameter passing register
HtoA    CMPI.B     #9, D5
        BGT        ABCD
NUM     ADDI.B     #$30, D5
        RTS
ABCD    ADDI.B     #$37, D5
        RTS
;======================================
```

Code Structure

- Overall Code
  - Start with A2H code
  - Use A2H for writing 4 hex bytes into MEM
  - Add new lines for retrieving and printing them



Code Run Example



Exercise IV

⌘DEC to HEX Conversion

```
** 3-Digit DEC to 3-Digit HEX Conversion **

TYPE A 3-Digit DEC number:
255
And the HEX equivalent is:
0FF

TYPE A 3-Digit DEC number:
120
And the HEX equivalent is:
078

TYPE A 3-Digit DEC number:
012
And the HEX equivalent is:
00C

TYPE A 3-Digit DEC number:
```

DEC to HEX Conversion – Background

```
DECIMAL      to    HEX         conversion(DtoH)
(3Digit)           (3Digit)
```

```
 125 =1x100 + 2x10 + 5
     =$(1x64 + 2x0A +5)
     =$(64+14+5)
     =$7D

1 --> $31 --> (AtoH) --> $01 -->($01x$64)-->$64
2 --> $32 --> (AtoH) --> $02 -->($02x$0A)-->$14
5 --> $35 --> (AtoH) --> $05 -------------->$05
                                $64+$14+$05-->$7D

                          $7D --> (HtoA)-->'7' & 'D'
```

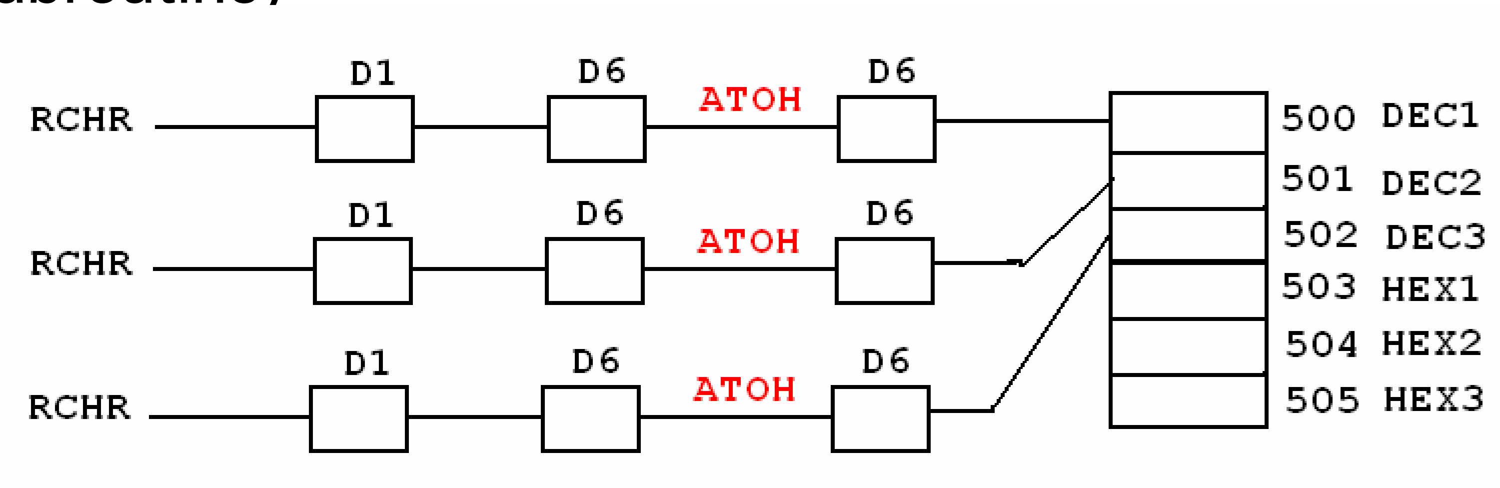
```
256=2*100 + 5*10 +6        How about this case?
   =$(2*64+5*0A+6)
   =$100
                          $02 x $64 --> $C8
                          $04 x $0A --> $28
                          $06       --> $06
                          -----------------
                               SUM-->$100

                               --> (HTOA)-->'1' '0'  '0'
```

Pre-Processing

- Declaration of Memory Location by Label

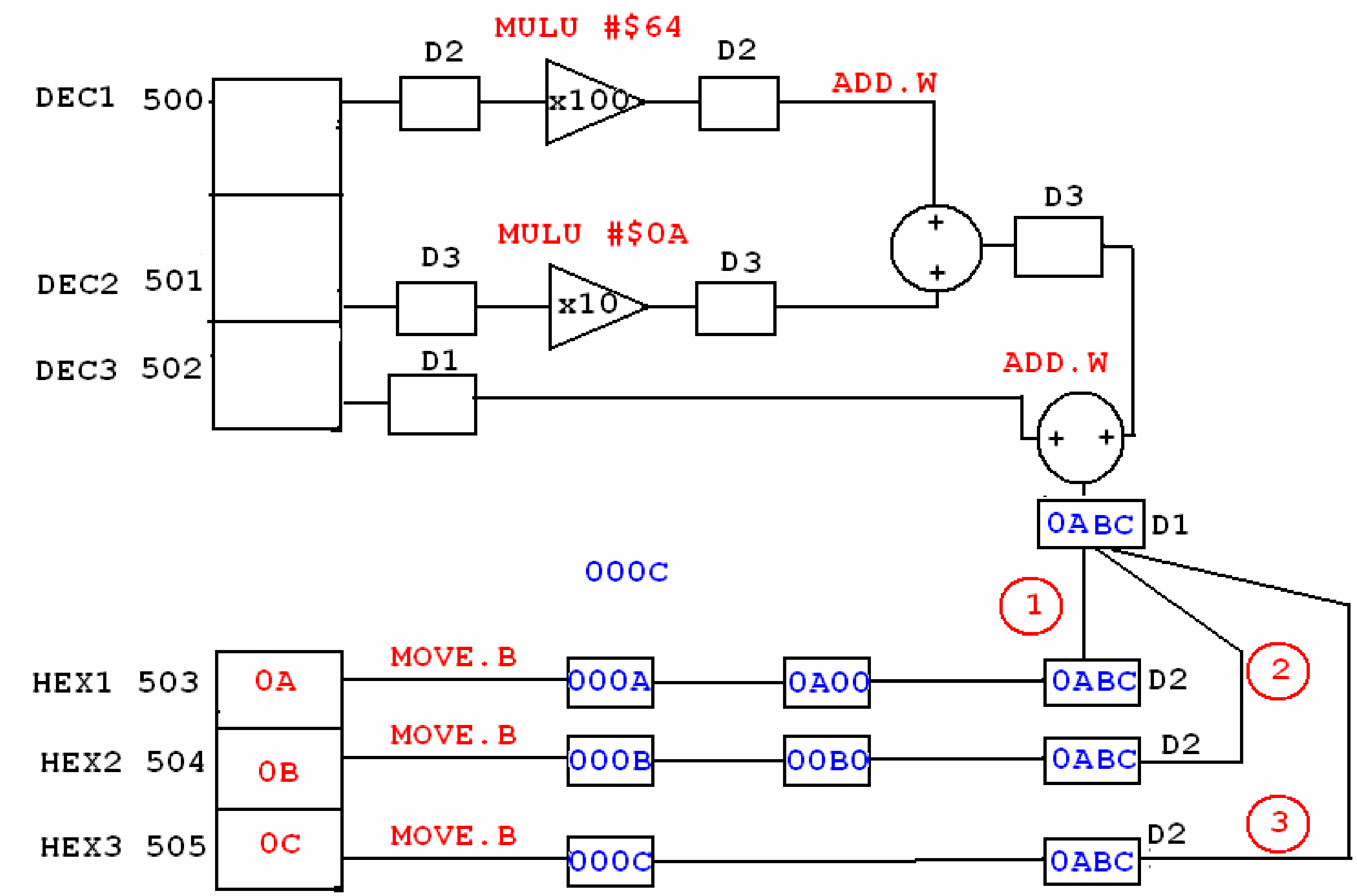
```
        ORG     $500
DEC1    DS.B    1
DEC2    DS.B    1
DEC3    DS.B    1
HEX1    DS.B    1
HEX2    DS.B    1
HEX3    DS.B    1
```

- Read 3 digit Decimal Number
- Store each digit from $500 as Number (by calling AtoH subroutine)



DtoH - Subroutine

- Read 3 numbers starting from $500 and store into Data Registers
- Convert them into 3 hex digits
- Store hex bytes starting from $503

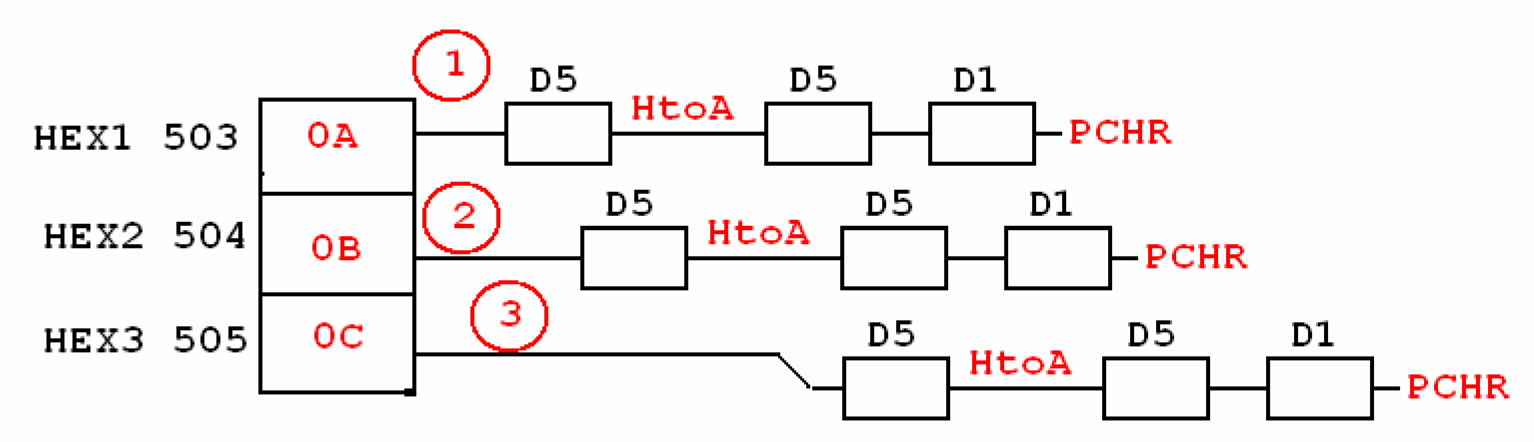


DtoH (subroutine code)

```
;D2H Subroutine
;INPUT: 3 digit dec number
;OUTPUT: 3 digit hex number
;Requirement 1: INPUT must be in DEC1 DEC2 DEC3 memory location
;Requirement 2: D1 D2 D3 are used for this subroutine
;Result: Output hex will be stored at HEX1 HEX2 HEX3 memory location'
DTOH    CLR.L   D2
        CLR.L   D3
        CLR.L   D1
        MOVE.B  DEC1, D2         ;(ex)823 = 8*100 + 2*10 +3
        MULU    #$64, D2
        MOVE.B  DEC2, D3
        MULU    #$0A, D3
        ADD.W   D2,D3
        MOVE.B  DEC3, D1
        ADD.W   D3,D1

        CLR.L   D2
        MOVE.W  D1,D2
        ANDI.W  #$0F00, D2
        LSR.W   #8, D2
        MOVE.B  D2, HEX1
        CLR.L   D2
        MOVE.W  D1, D2
        ANDI.W  #$00F0, D2
        LSR.W   #4,D2
        MOVE.B  D2, HEX2
        CLR.L   D2
        MOVE.B  D1, D2
        ANDI.B  #$0F, D2
        MOVE.B  D2, HEX3
        RTS
```

The Last Step & Overall

- The last step:
  - Read each Hex byte starting from $503
  - Convert each Hex to ASCII (by calling HtoA subroutine) then Display (by PCHR)



- Overall Structure
  - 1. Pre-Processing
  - 2. Call DtoH subroutine
  - 3. Last Step

Run Result and Memory Contents

Sim68K I/O

```
+++3-DIGIT DEC to 3-DIGIT HEX CONVERSION +++

TYPE A 3-Digit DEC number: 973
And the HEX equivalent is: 3CD

TYPE A 3-Digit DEC number:
```

```
00000490: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000004F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000500: 09 07 03 03 0C 0D FF FF FF FF FF FF FF FF FF FF ----------------
00000510: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000520: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
```

Live

68K Coding Project

- **Coding Problem**:
  - Convert the keyed-in “Dotted Decimal IP address” into “8-digit HEX number” and display the number AND its class (A – E)
  - The number of digits of each decimal number can be 1, 2, or 3.
  - The last decimal number does not have ‘dot’
  - The ‘Enter’ key indicates the end of the IP address input.
- **Submission Instruction**
  - Overall code structure in the register and memory level details.
  - Code with plenty of comments (almost every line of instruction)
  - Deadline: TBD

IP Address & Result Display Example

LAN IP Address Classification

Class A = 1.0.0.0 to 127.255.255.255
Class B = 128.0.0.0 to 191.255.255.255
Class C = 192.0.0.0 to 223.255.255.255
Class D = 224.0.0.0 to 247.255.255.255

IP Address Formatting

Class	Byte 1	Byte 2	Byte 3	Byte 4
A	Network	Host		
B	Network		Host	
C	Network			Host
D	Multicast			

138.238.121.68 ↴

8A:EE:79:44

B

138.238.10.7 ↴

8A:EE:0A:07

B