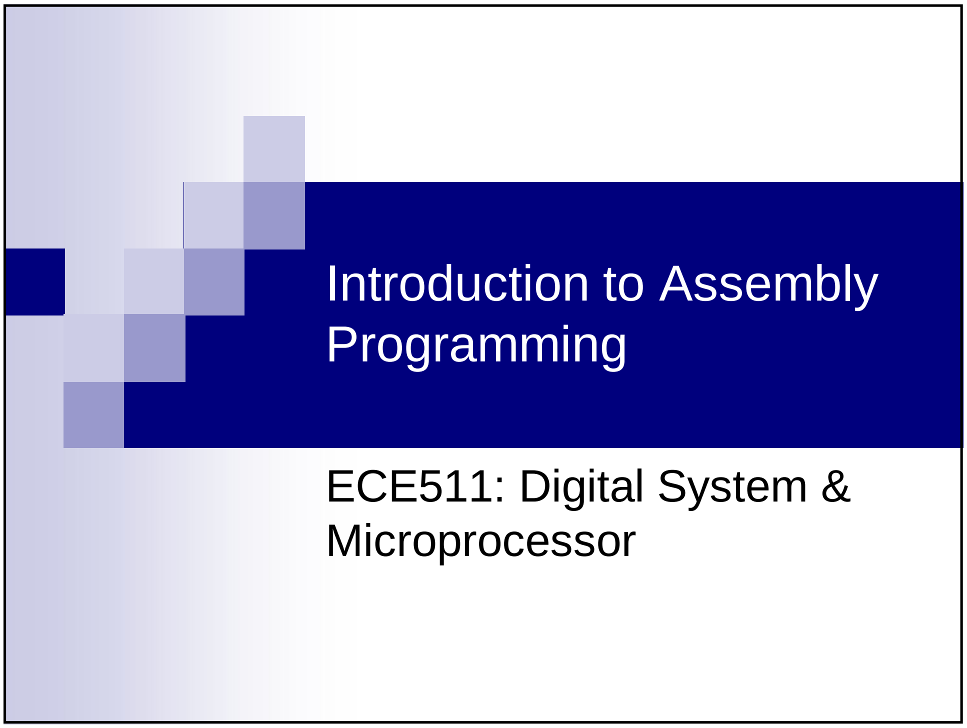
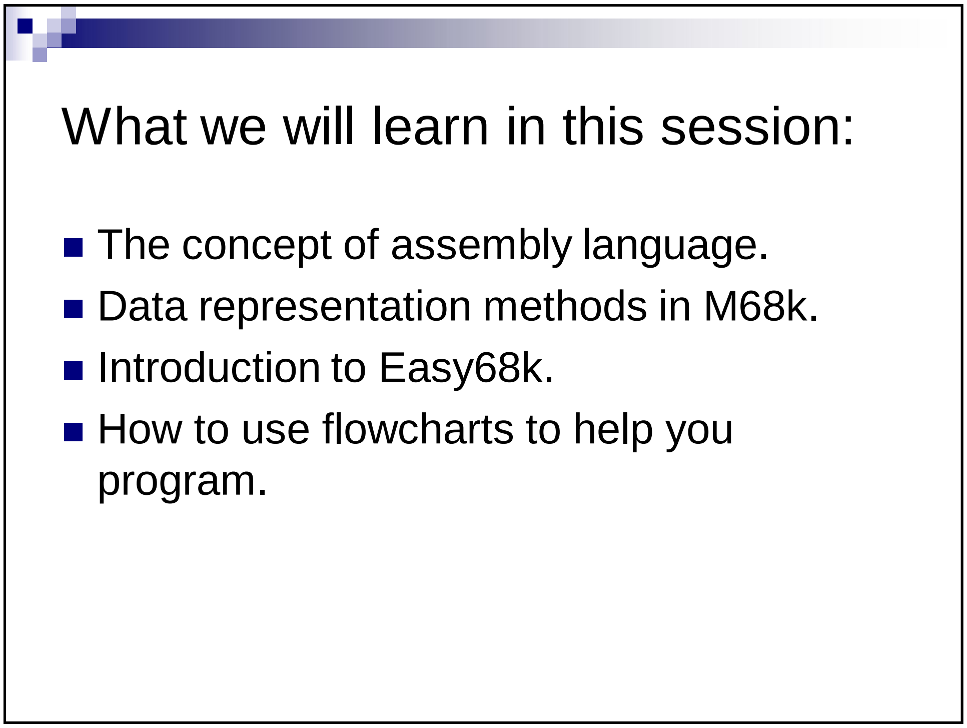


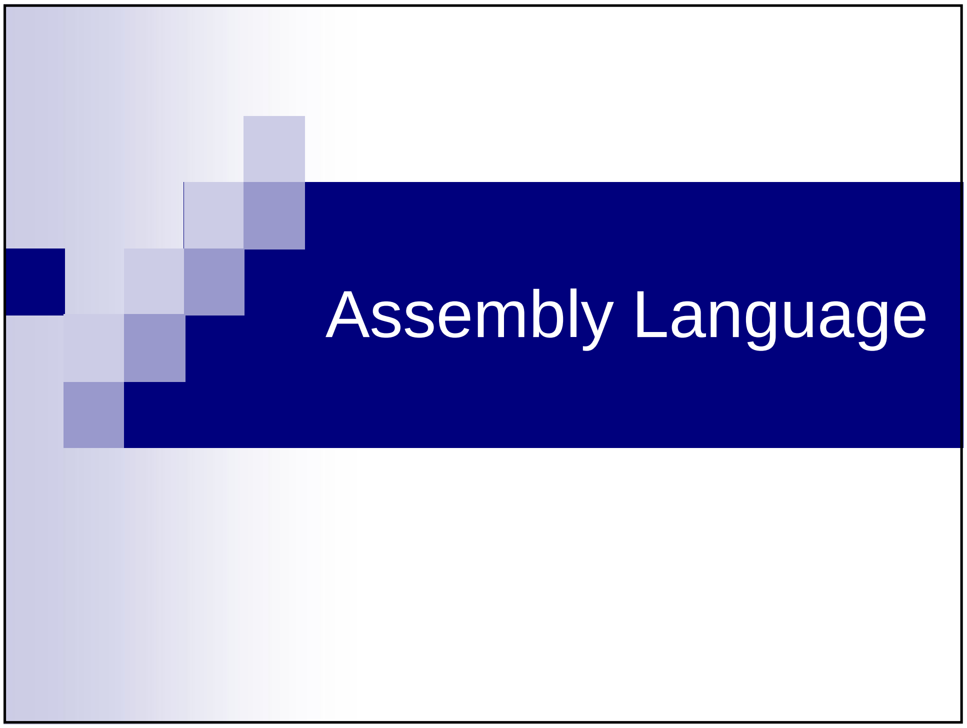
Introduction to Assembly Programming

ECE511: Digital System & Microprocessor

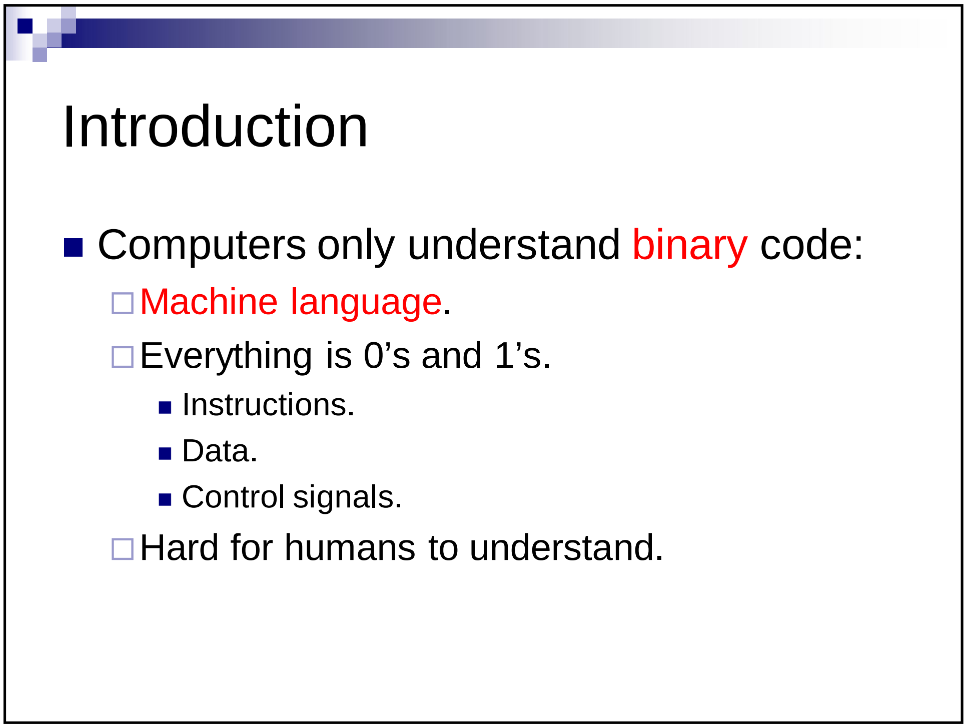


What we will learn in this session:

- The concept of assembly language.
- Data representation methods in M68k.
- Introduction to Easy68k.
- How to use flowcharts to help you program.



Assembly Language



Introduction

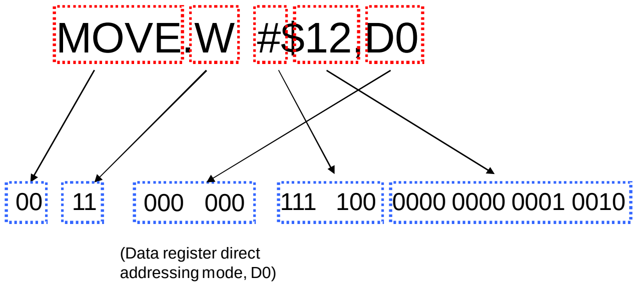
- Computers only understand binary code:
  - □ Machine language.
  - □ Everything is 0’s and 1’s.
    - Instructions.
    - Data.
    - Control signals.
  - □ Hard for humans to understand.

Try and translate this:

00110000001111000000000000010010

Try and translate this:

00110000001111000000000000010010



MOVE.W #$12,D0

00 11 000 000 111 100 0000 0000 0001 0010

(Data register direct addressing mode, D0)

Assembly Language

- We can represent machine code in Assembly Language:
  - □ Easier to understand, program.
- Simple, low-level programming language.
  - □ Using mnemonics (ADD, MOVE, MULU).
  - □ Close to human language.
- Code generated is machine-specific:
  - □ Each μP has own assembly language.

Assembly Language

- Assembler to generate machine code.
  - □ Object file (Motorola: S-file).
  - □ Contains machine code.
- Linker sometimes used for big projects:
  - □ Links together multiple S-files.
  - □ Creates single S-file from combined files.

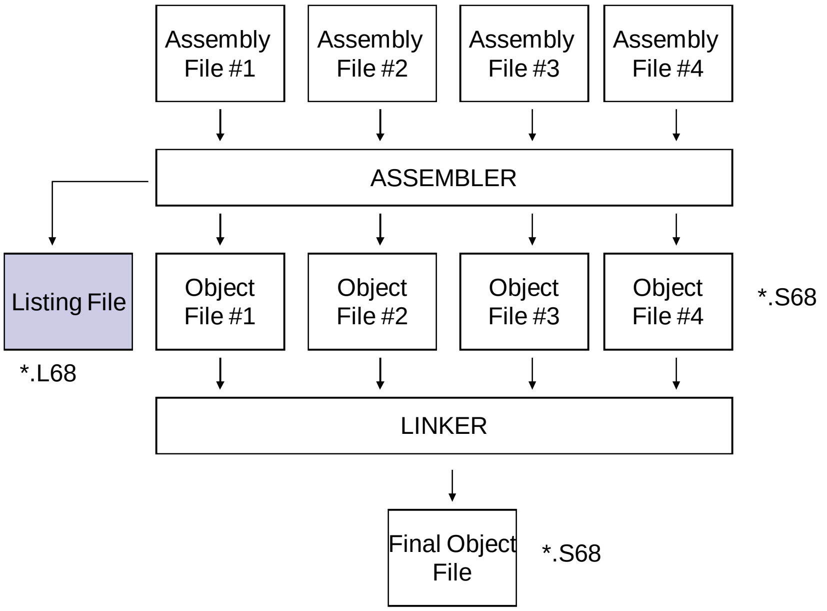
The Assembler

- Assembly Language:
  - □ MOVE.W #$12,D0
  - □ MOVE.W #$34,D1
  - □ MULU D1,D0

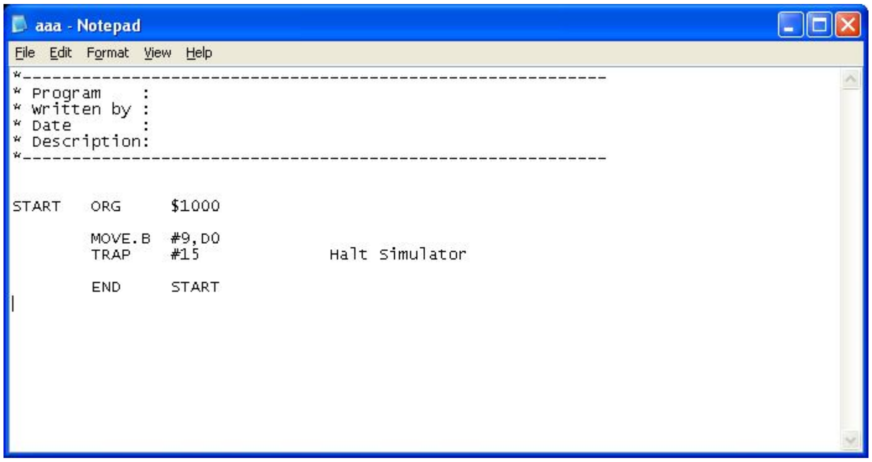
Assembler

- Machine Language:

00110000001111000
01100100011110000
00000000110100

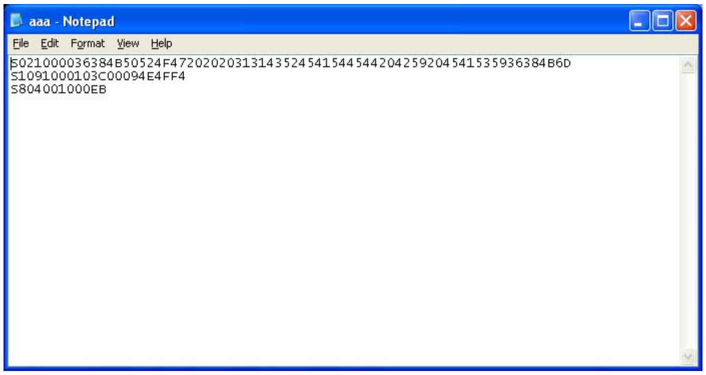


Source Code (X68)



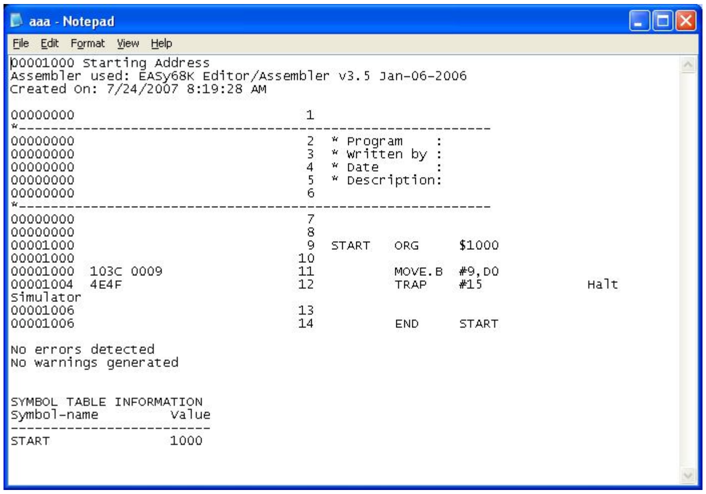
```
*-----------------------------------------------------------
* Program    :
* Written by :
* Date       :
* Description:
*-----------------------------------------------------------


START   ORG     $1000

        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     START
```

Object File (S-File)



```
S021000036384B50524F47202020313143524541544544204259204541535936384B6D
S1091000103C00094E4FF4
S804001000EB
```

Listing File



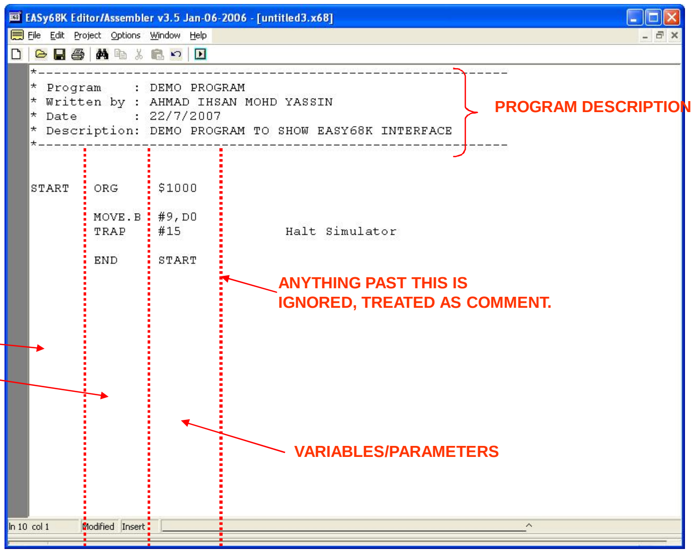
```
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 7/24/2007 8:19:28 AM

00000000                             1
*-----------------------------------------------------------
00000000                             2  * Program    :
00000000                             3  * Written by :
00000000                             4  * Date       :
00000000                             5  * Description:
00000000                             6
*-----------------------------------------------------------
00000000                             7
00000000                             8
00001000                             9  START   ORG     $1000
00001000                            10
00001000  103C 0009                 11          MOVE.B  #9,D0
00001004  4E4F                      12          TRAP    #15             Halt
Simulator
00001006                            13
00001006                            14          END     START

No errors detected
No warnings generated


SYMBOL TABLE INFORMATION
Symbol-name         Value
-------------------------
START               1000
```

Easy68k

Easy68k

- Designed by Prof. C. Kelly, Monroe County Community College.
- Freeware.
- Installer: http://www.monroeccc.edu/ckelly/Files/SetupEASy68K.exe
- Easy68k Quick-Ref: http://www.monroeccc.edu/ckelly/easy68k/EASy68KQuickRefv1_8.pdf

Easy68k Interface


EASy68K Editor/Assembler v3.5 Jan-06-2006 - [untitled3.x68]
File Edit Project Options Window Help
ln 10 col 1
Modified
Insert

```
*-----------------------------------------------------------
* Program     : DEMO PROGRAM
* Written by  : AHMAD IHSAN MOHD YASSIN
* Date        : 22/7/2007
* Description: DEMO PROGRAM TO SHOW EASY68K INTERFACE
*-----------------------------------------------------------

START   ORG     $1000

        MOVE.B  #9,D0
        TRAP    #15             Halt Simulator

        END     START
```

PROGRAM DESCRIPTION

ANYTHING PAST THIS IS IGNORED, TREATED AS COMMENT.

LABEL

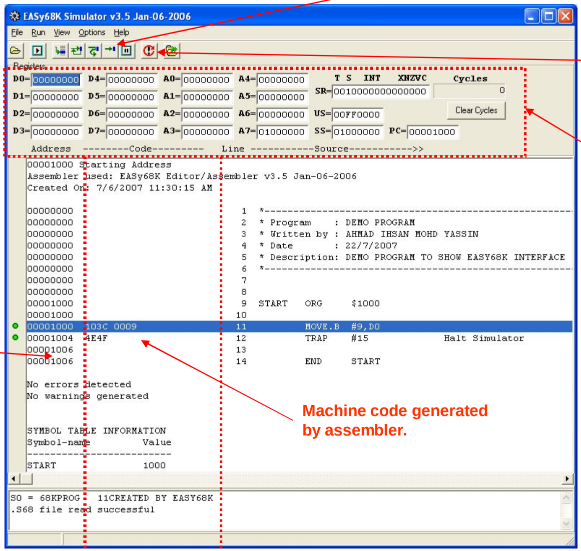
INSTRUCTION

VARIABLES/PARAMETERS

Programming in Easy 68k

- Easy68k divides program into columns:
  - □ Label:
    - Marks memory locations using characters.
    - Easy reference.
  - □ Instruction:
    - What instruction to execute.
  - □ Variables/Parameters:
    - Usually specifies source & destination.
    - May specify parameters as well.
  - □ Comment:
    - Used to describe flow of program,

Simulation Window


EASy68K Simulator v3.5 Jan-06-2006
File Run View Options Help
Registers
Clear Cycles

```
D0=00000000  D4=00000000  A0=00000000  A4=00000000     T S  INT  XNZVC     Cycles
D1=00000000  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000000         0
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001000
   Address   --------Code---------   Line ----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 7/6/2007 11:30:15 AM

00000000                               1  *-------------------------------------------------
00000000                               2  * Program     : DEMO PROGRAM
00000000                               3  * Written by  : AHMAD IHSAN MOHD YASSIN
00000000                               4  * Date        : 22/7/2007
00000000                               5  * Description: DEMO PROGRAM TO SHOW EASY68K INTERFACE
00000000                               6  *-------------------------------------------------
00000000                               7
00000000                               8
00001000                               9  START   ORG     $1000
00001000                              10
00001000  103C 0009                   11          MOVE.B  #9,D0
00001004  4E4F                        12          TRAP    #15             Halt Simulator
00001006                              13
00001006                              14          END     START

No errors detected
No warnings generated


SYMBOL TABLE INFORMATION
Symbol-name          Value
-------------------------
START                1000
```

```
S0 = 68KPROG   11CREATED BY EASY68K
.S68 file read successful
```

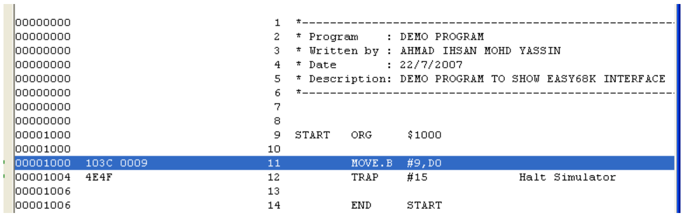
Press here to execute programs step-by-step.

Press here to restart program execution.

Shows status of internal M68k registers.

Memory address where instruction is stored.

Machine code generated by assembler.



```
00000000                        1  *-----------------------------------------------------
00000000                        2  * Program    : DEMO PROGRAM
00000000                        3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                        4  * Date       : 22/7/2007
00000000                        5  * Description: DEMO PROGRAM TO SHOW EASY68K INTERFACE
00000000                        6  *-----------------------------------------------------
00000000                        7
00000000                        8
00001000                        9  START   ORG     $1000
00001000                       10
00001000  103C 0009            11          MOVE.B  #9,D0
00001004  4E4F                 12          TRAP    #15             Halt Simulator
00001006                       13
00001006                       14          END     START
```

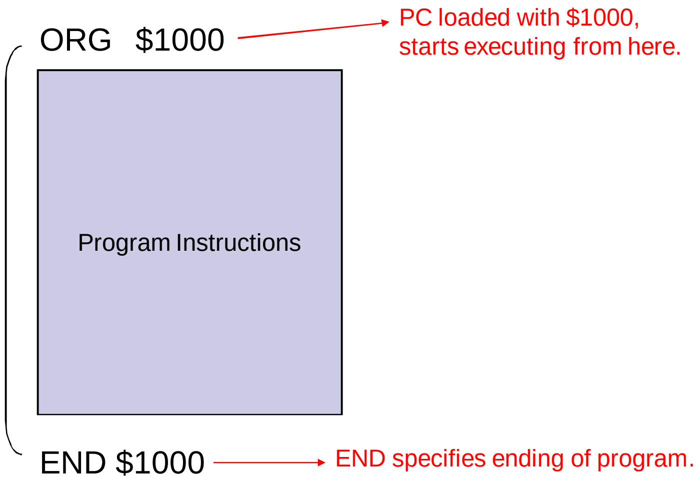
MOVE.B #9,D0 instruction:

Is on line 11 in M68k source file.
Is stored in memory address $1000.
Its machine code is $103C0009 (00010000001111000000000000001001).

Specify Start/End of Program

- M68k needs to know where to start executing instructions, where to stop.
- Specified using ORG (Origin), END (End).
- Value of ORG loaded into PC, execution starts there.

Format: Start/End Program



ORG $1000

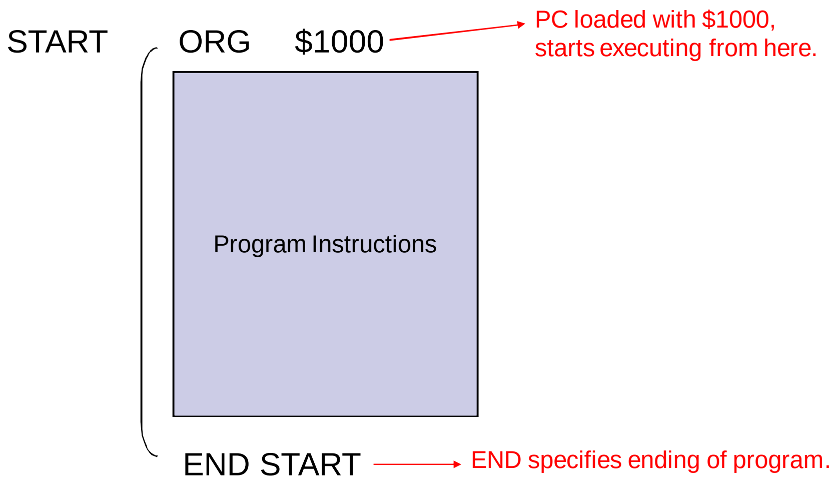
PC loaded with $1000, starts executing from here.

Program Instructions

END $1000

END specifies ending of program.

Format: Start/End Program



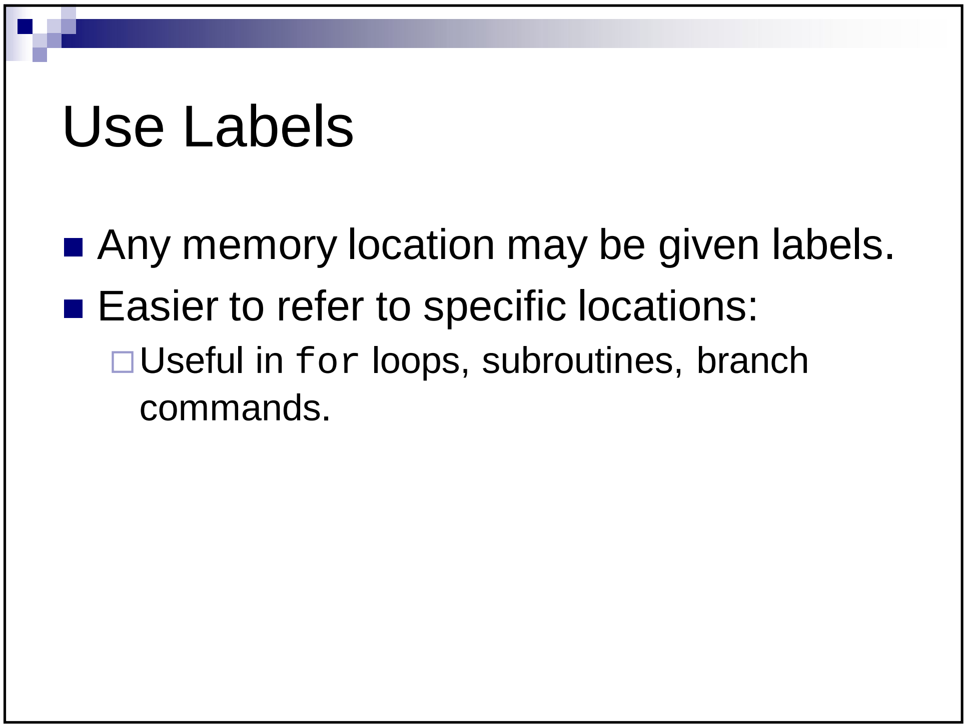
START ORG $1000

PC loaded with $1000, starts executing from here.

Program Instructions

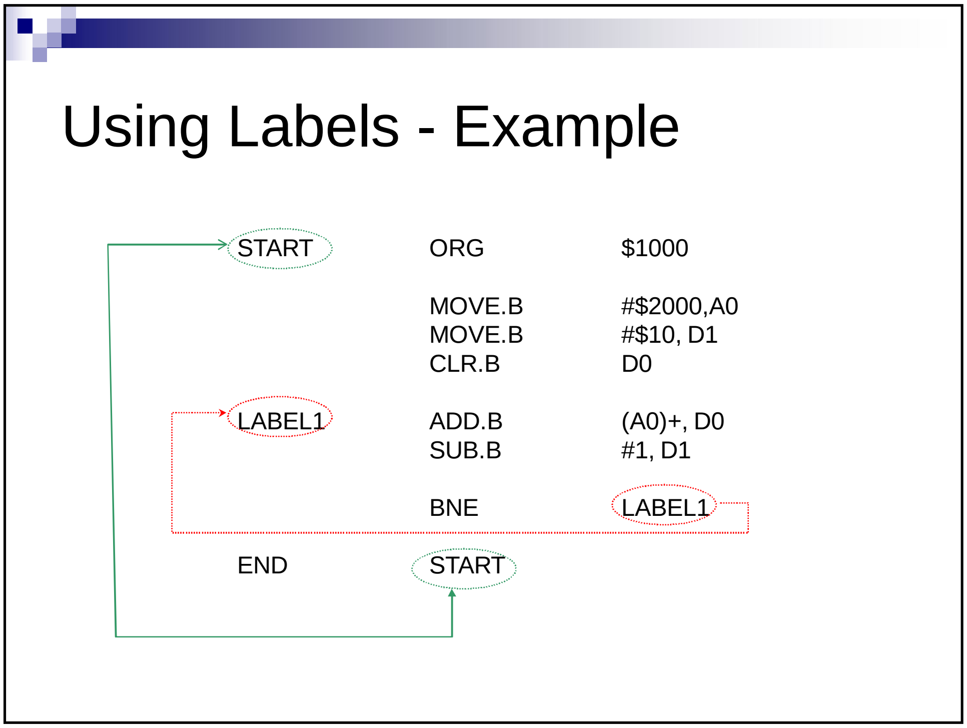
END START

END specifies ending of program.



Use Labels

- Any memory location may be given labels.
- Easier to refer to specific locations:
  - Useful in `for` loops, subroutines, branch commands.



Using Labels - Example

```
START    ORG        $1000

         MOVE.B     #$2000,A0
         MOVE.B     #$10, D1
         CLR.B      D0

LABEL1   ADD.B      (A0)+, D0
         SUB.B      #1, D1

         BNE        LABEL1

END      START
```

Data Representation Methods

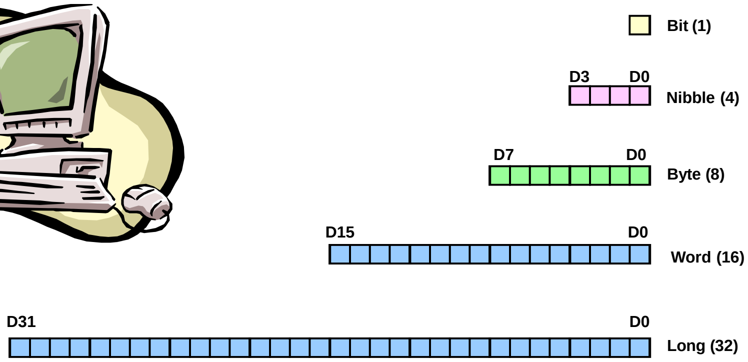
Data Sizes

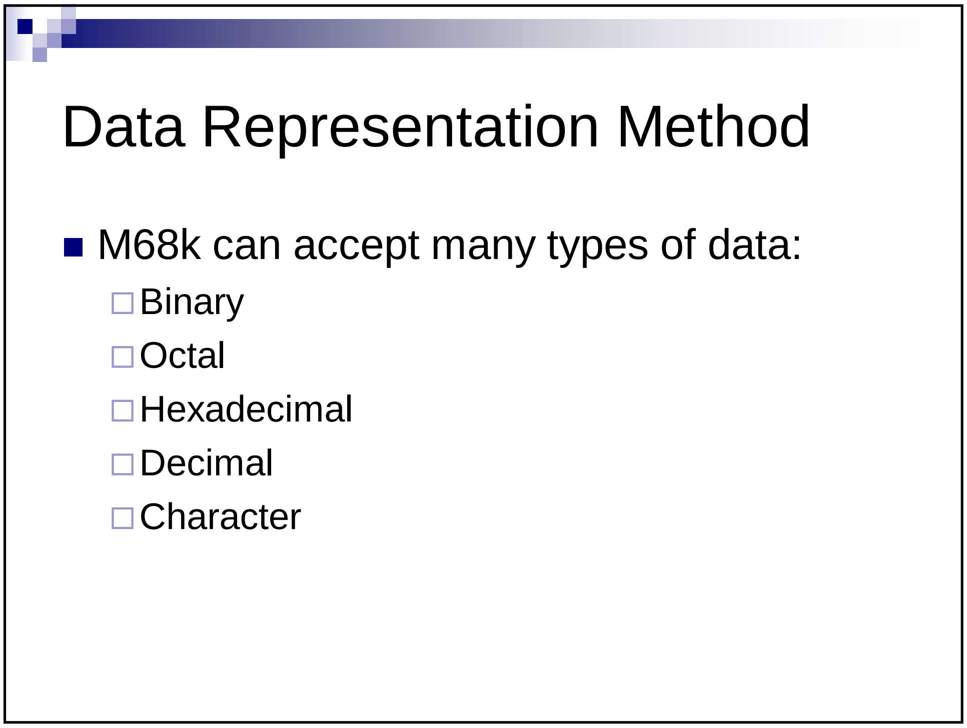
- Bit:
  - Most basic representation.
  - Contains either 0 or 1.
  - Can be grouped together to represent more meaning.
- Nibble: 4 bits.
  - Can represent 16 values (2^4).
  - Not recognized in M68k.
  - Need to write special program to handle.
- Byte: 8 bits.
  - Indicated by ".B" notation.
  - Can hold value up to 256 (2^8).

Data Sizes

- Word: 16 bits.
  - □ Length of most instructions in M68k.
  - □ Can hold value up to 65,535 (2^{16}).
  - □ Indicated by “.W” notation.
- Long: 32 bits.
  - □ Length of data registers in M68k.
  - □ Can hold value up to 4,294,967,296 (2^{32}).
  - □ Indicated by “.L” notation.

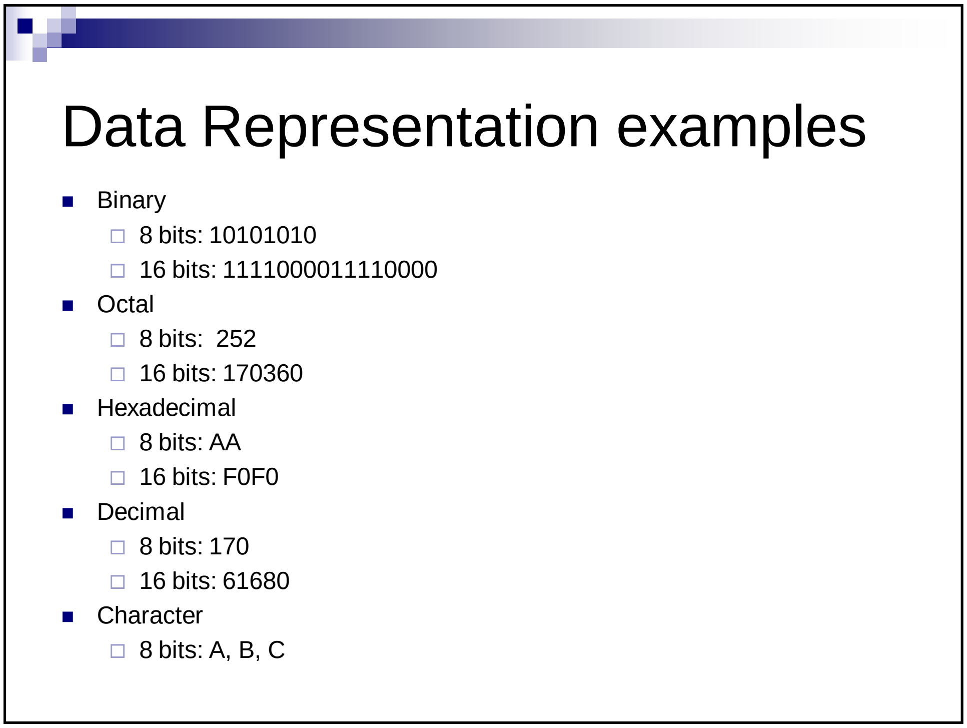
Data Sizes





Data Representation Method

- M68k can accept many types of data:
  - Binary
  - Octal
  - Hexadecimal
  - Decimal
  - Character

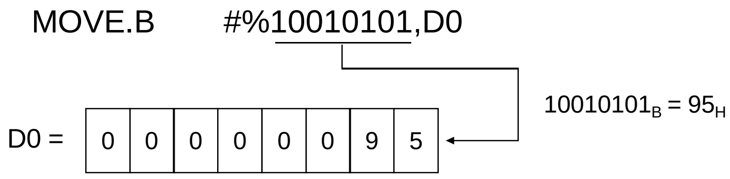


Data Representation examples

- Binary
  - 8 bits: 10101010
  - 16 bits: 1111000011110000
- Octal
  - 8 bits: 252
  - 16 bits: 170360
- Hexadecimal
  - 8 bits: AA
  - 16 bits: F0F0
- Decimal
  - 8 bits: 170
  - 16 bits: 61680
- Character
  - 8 bits: A, B, C

Binary

- Binary: start with %
- Example:
  - □ Move binary value 10010101 to D0.



```
MOVE.B      #%10010101,D0
```

D0 =

0	0	0	0	0	0	9	5

$10010101_B = 95_H$

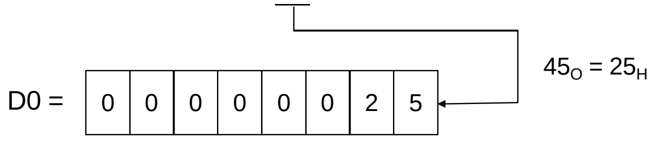
Binary

- MOVE.B #%10110110, D0
- MOVE.B #%100010111, D0

Octal

- Octal: start with @
- Example:
  - □Move octal value 45 to D0.

```
MOVE.B      #@45,D0
```



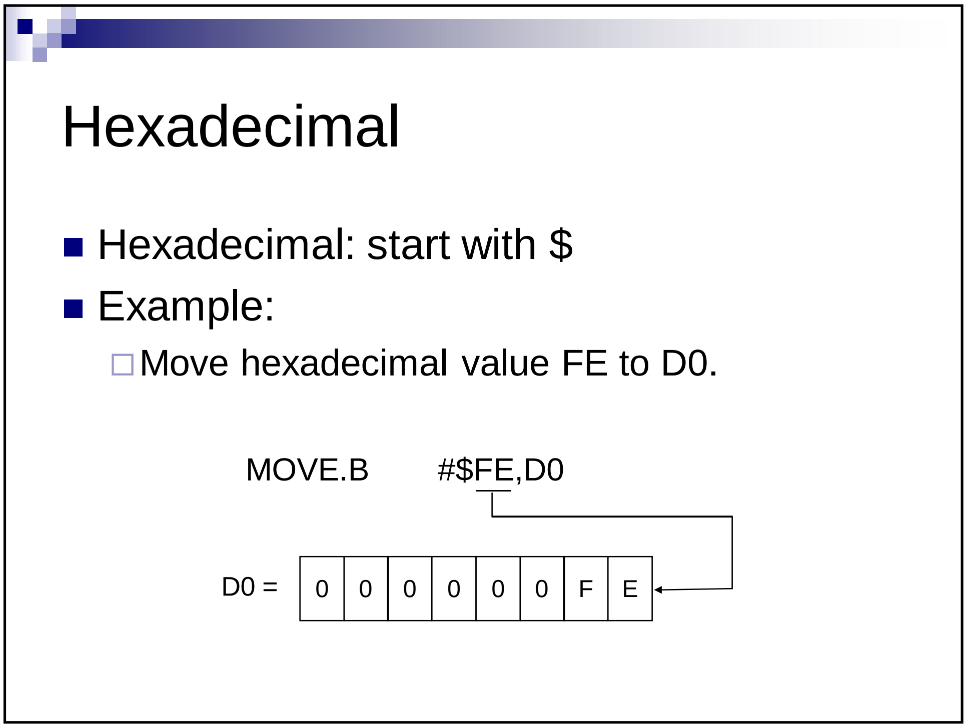
D0 =

0	0	0	0	0	0	2	5

$45_O = 25_H$

Octal

- MOVE.B   #@377, D0
- MOVE.B   #@400, D0

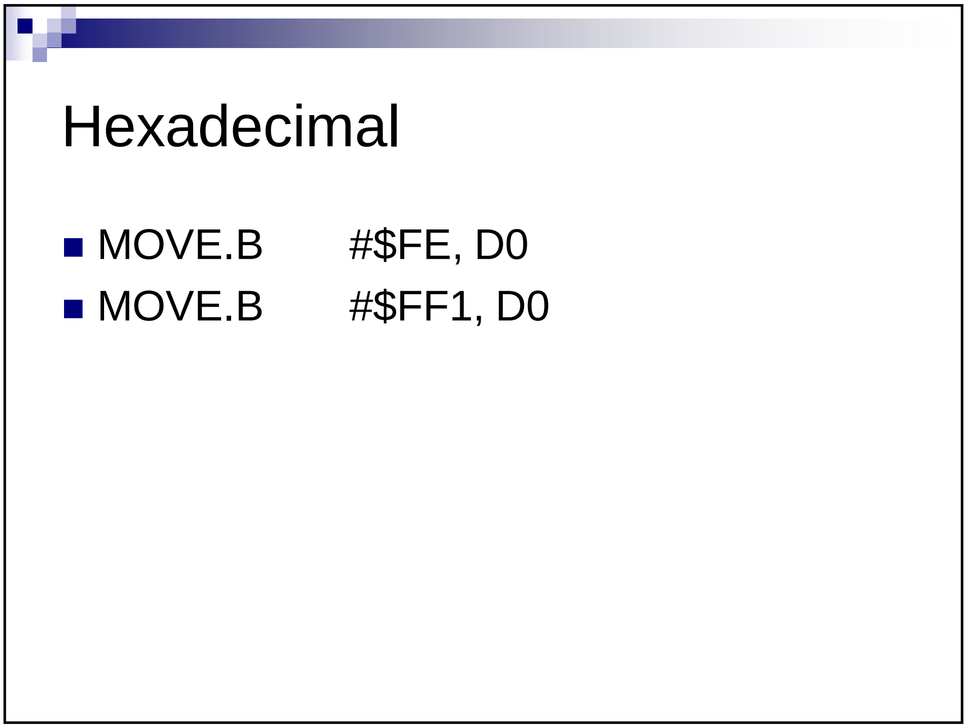


Hexadecimal

- Hexadecimal: start with $
- Example:
  - □ Move hexadecimal value FE to D0.

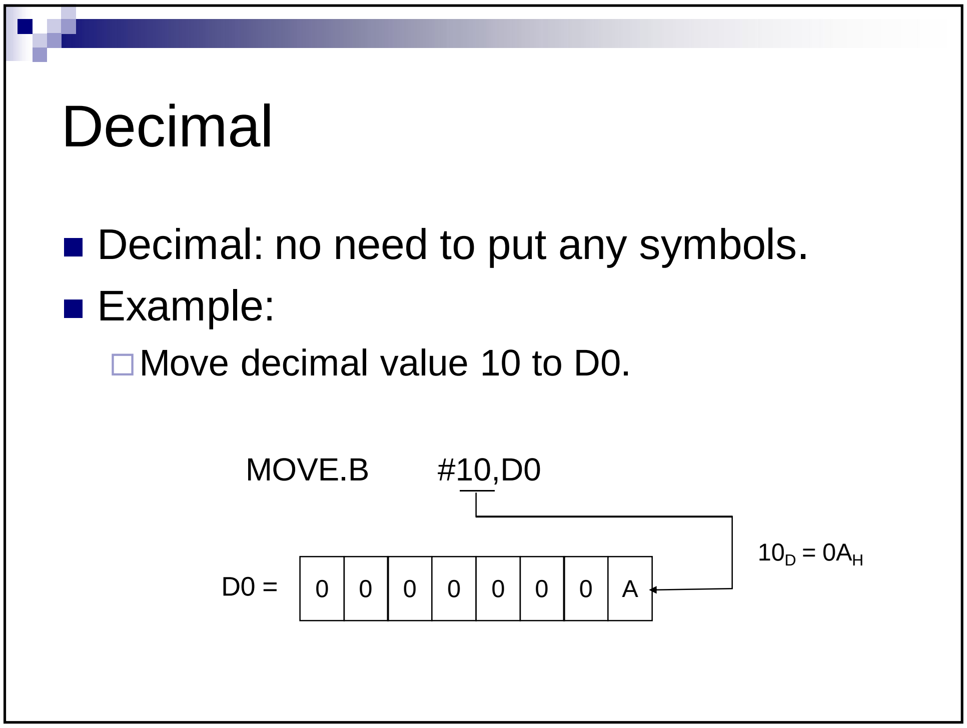
```
MOVE.B        #$FE,D0
```

D0 =	0	0	0	0	0	0	F	E



Hexadecimal

- MOVE.B   #$FE, D0
- MOVE.B   #$FF1, D0



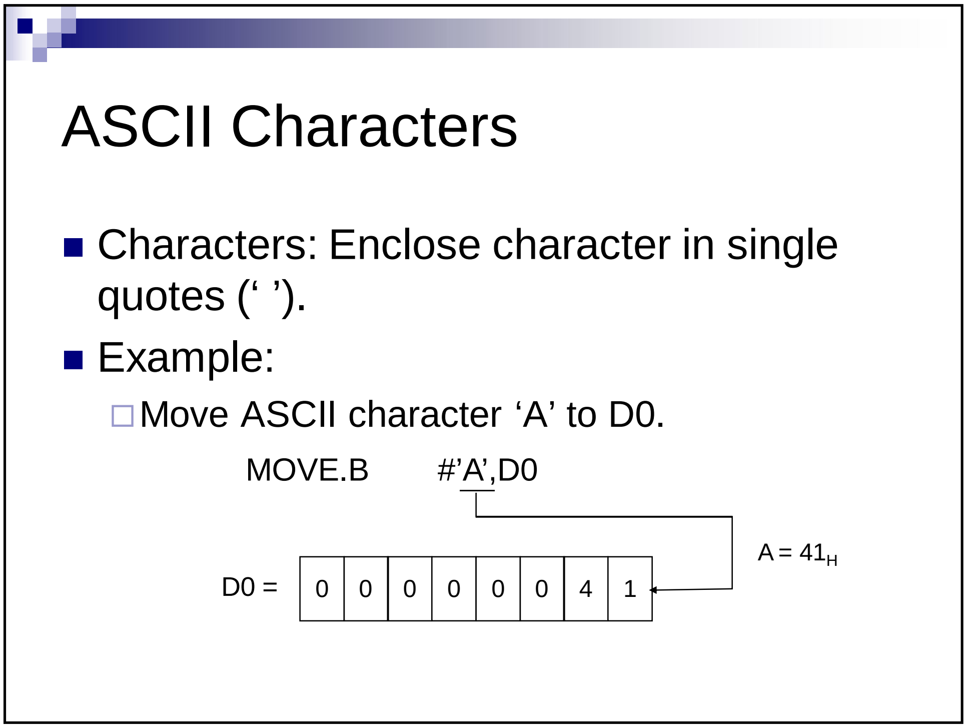
Decimal

- Decimal: no need to put any symbols.
- Example:
  - □Move decimal value 10 to D0.

```
MOVE.B     #10,D0
```

$10_D = 0A_H$

D0 =	0	0	0	0	0	0	0	A



ASCII Characters

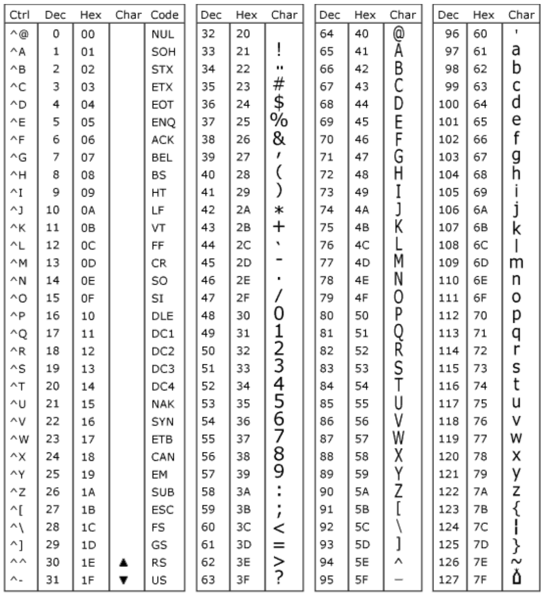
- Characters: Enclose character in single quotes (‘ ’).
- Example:
  - □Move ASCII character ‘A’ to D0.

```
MOVE.B     #’A’,D0
```

$A = 41_H$

D0 =	0	0	0	0	0	0	4	1

ASCII Characters



Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Flowchart

Flowchart

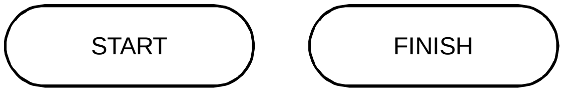
- Graphical method to plan flow of our programs.
- Shows program’s step-by-step operation.
- Easy to understand and analyze.
- Can be used to write organized programs.

Flowchart

- Basic shapes:
  - □ Terminator.
  - □ Process.
  - □ Decision.
  - □ Input/Output.
  - □ Connectors.

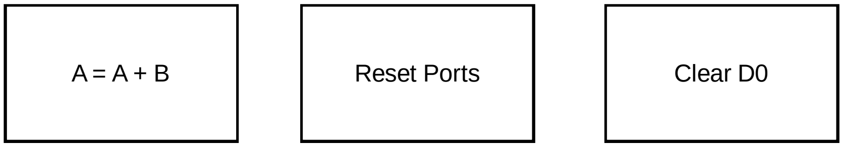
Basic Shapes – Terminator

- Indicates beginning and end of flowchart.
- Once at beginning, once at end.
- Examples:



Basic Shapes - Process

- Describes actions to be done.
- Represented as rectangles.
    - Short description of process in rectangle.
- Example:

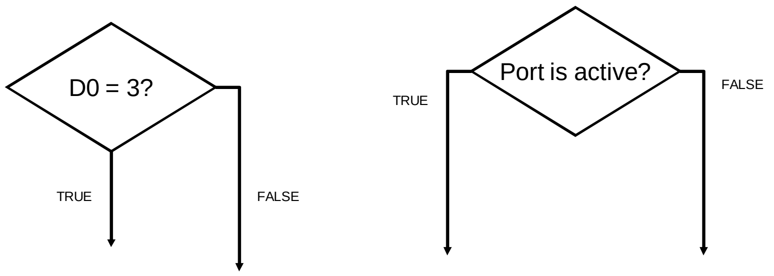


Basic Shapes - Decision

- Shows alternative program flow based on condition.
- Represented as diamond shape.
- Should have 2 arrows, representing TRUE and FALSE program flows.
- Can be used in “if…else”, “while”, and “for” situations.

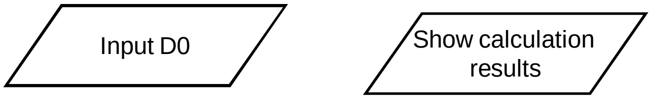
Basic Shapes - Decision

- Examples:



Basic Shapes – Input/Output

- Shows the process of inputting or outputting data.
- Represented using rhombus.
- Examples:



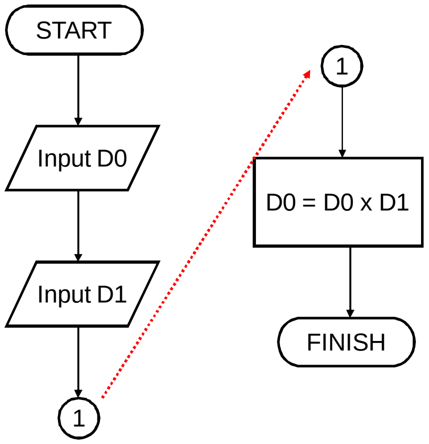
Input D0

Show calculation results

Basic Shapes - Connectors

- Used to link large process flows together.
- Represented using circles, with numbers inside.
- Numbers indicate connection.
- Examples:

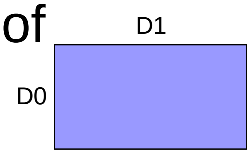
Example: Connector

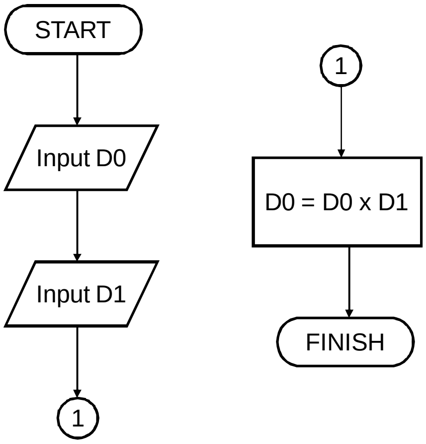


Writing the Program

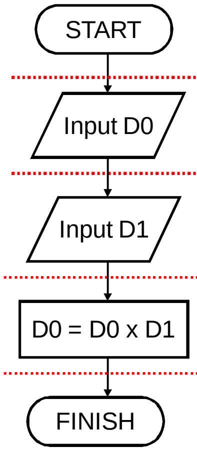
- Once flowchart is complete, write code to implement program.
- Follow program flow closely.
- Check and fix problems if necessary.

Example: Calculate Area of Rectangle

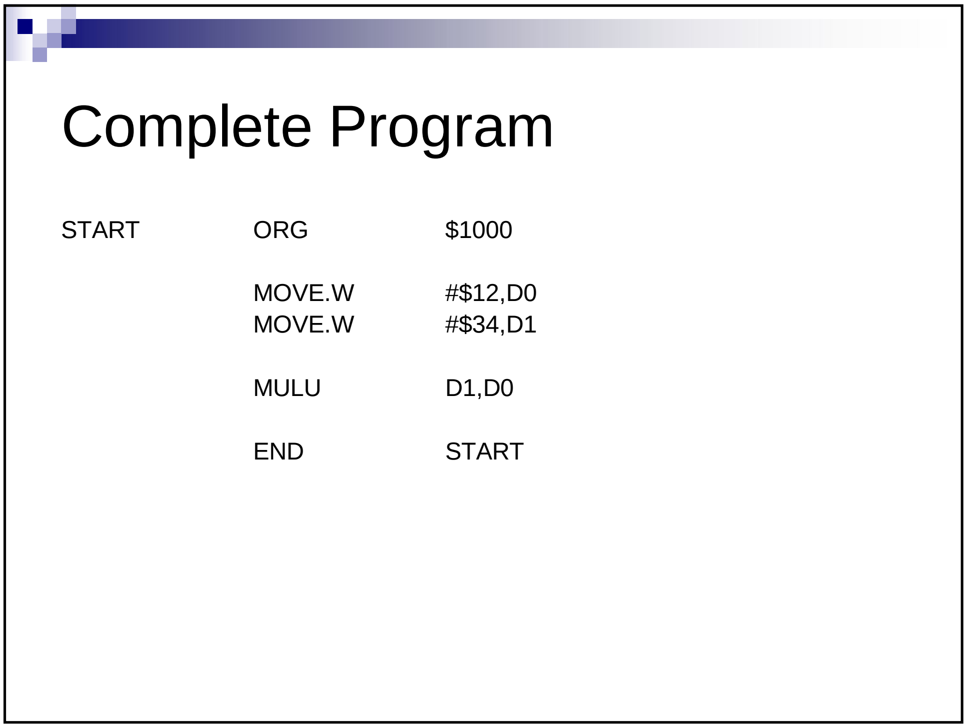




Translation to Assembly

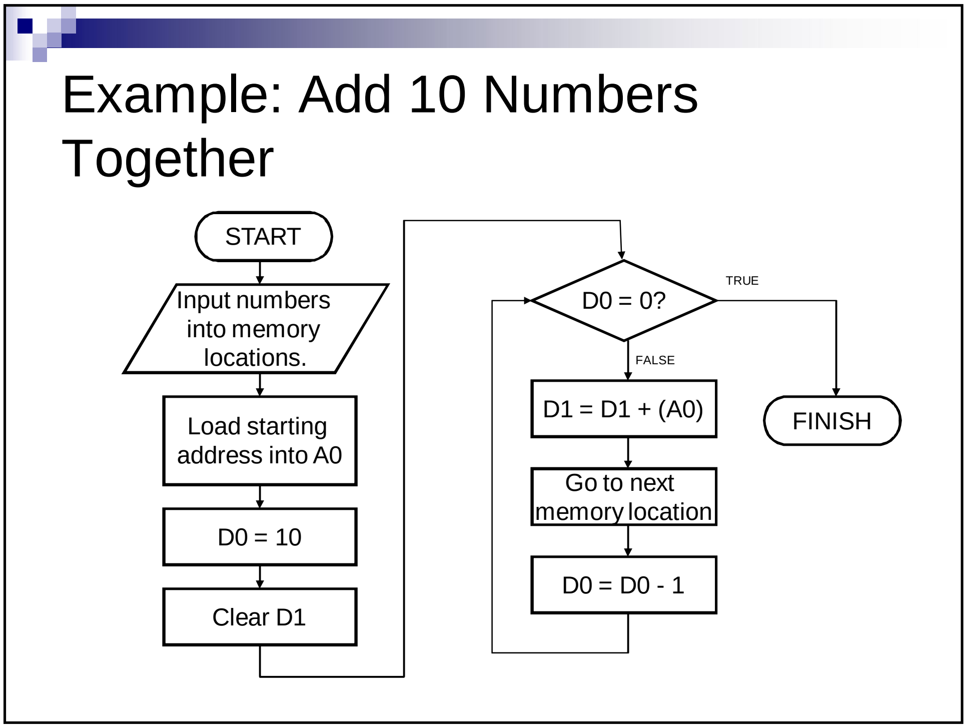


```
ORG       $1000
MOVE.W    #$12,D0
MOVE.W    #$34,D1
MULU      D1,D0
END
```



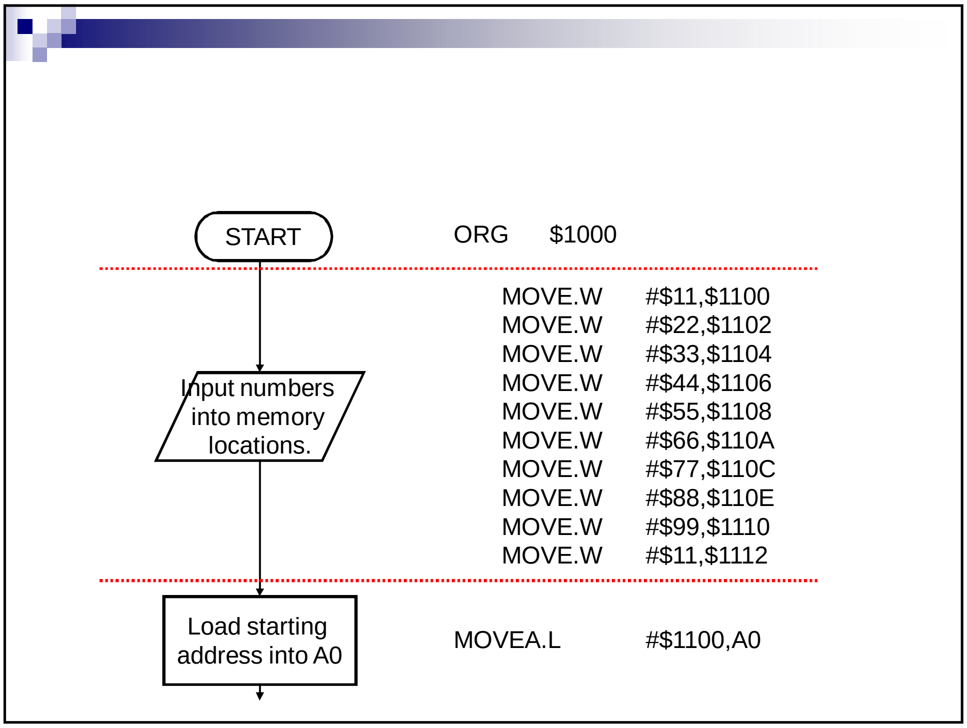
Complete Program

```
START     ORG       $1000

          MOVE.W    #$12,D0
          MOVE.W    #$34,D1

          MULU      D1,D0

          END       START
```



Example: Add 10 Numbers Together

START
Input numbers into memory locations.
Load starting address into A0
D0 = 10
Clear D1
D0 = 0?
TRUE
FALSE
D1 = D1 + (A0)
Go to next memory location
D0 = D0 - 1
FINISH

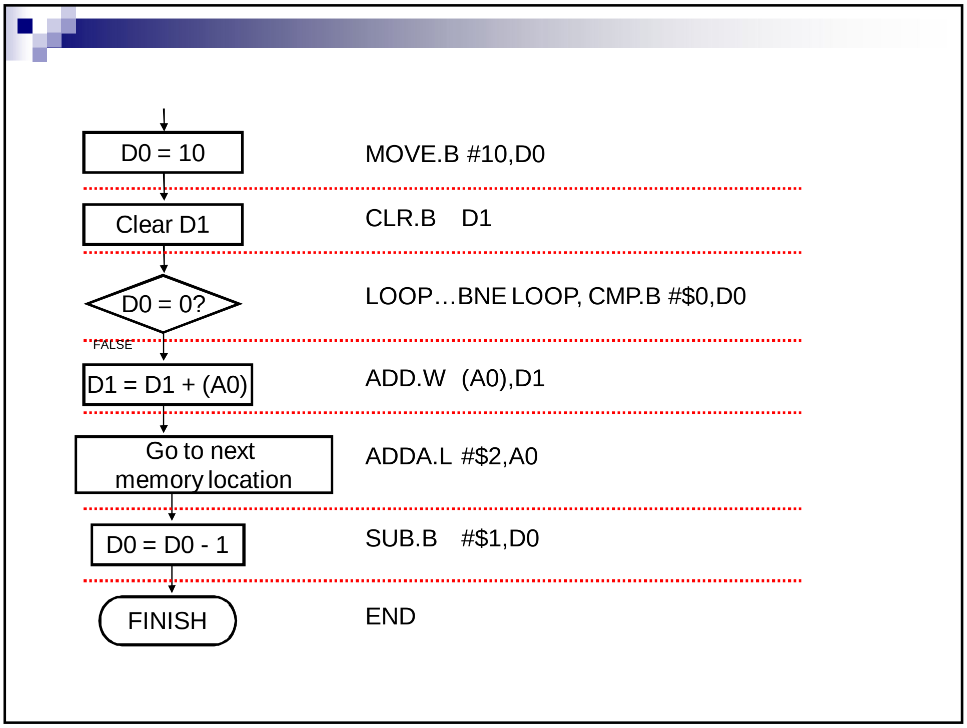


START

Input numbers into memory locations.

Load starting address into A0

```
ORG     $1000

    MOVE.W      #$11,$1100
    MOVE.W      #$22,$1102
    MOVE.W      #$33,$1104
    MOVE.W      #$44,$1106
    MOVE.W      #$55,$1108
    MOVE.W      #$66,$110A
    MOVE.W      #$77,$110C
    MOVE.W      #$88,$110E
    MOVE.W      #$99,$1110
    MOVE.W      #$11,$1112

MOVEA.L         #$1100,A0
```



D0 = 10

Clear D1

D0 = 0?

FALSE

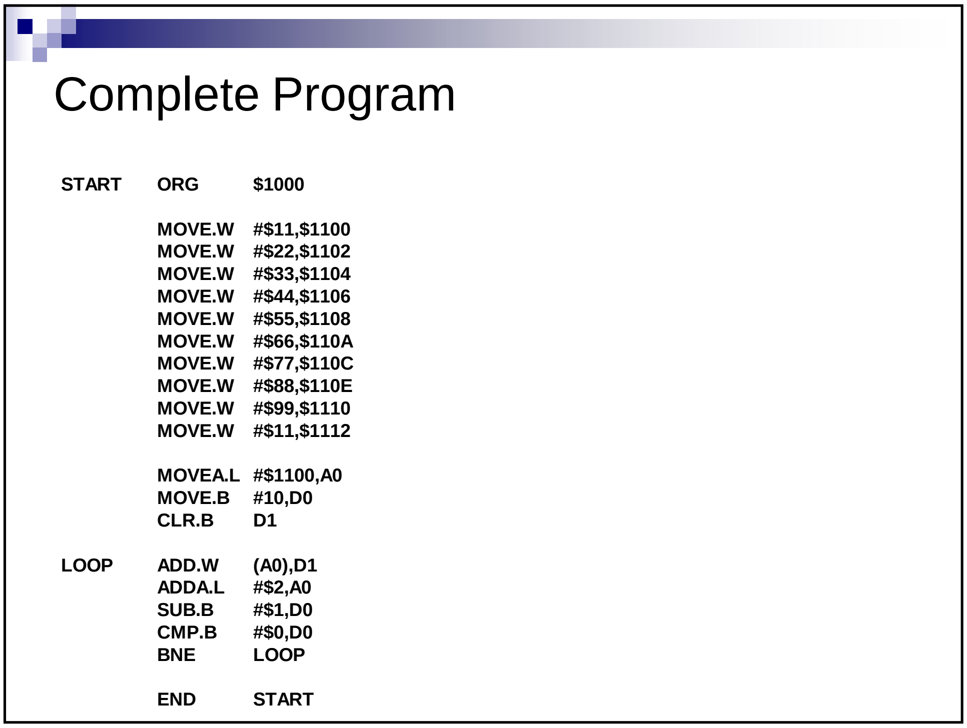
D1 = D1 + (A0)

Go to next memory location

D0 = D0 - 1

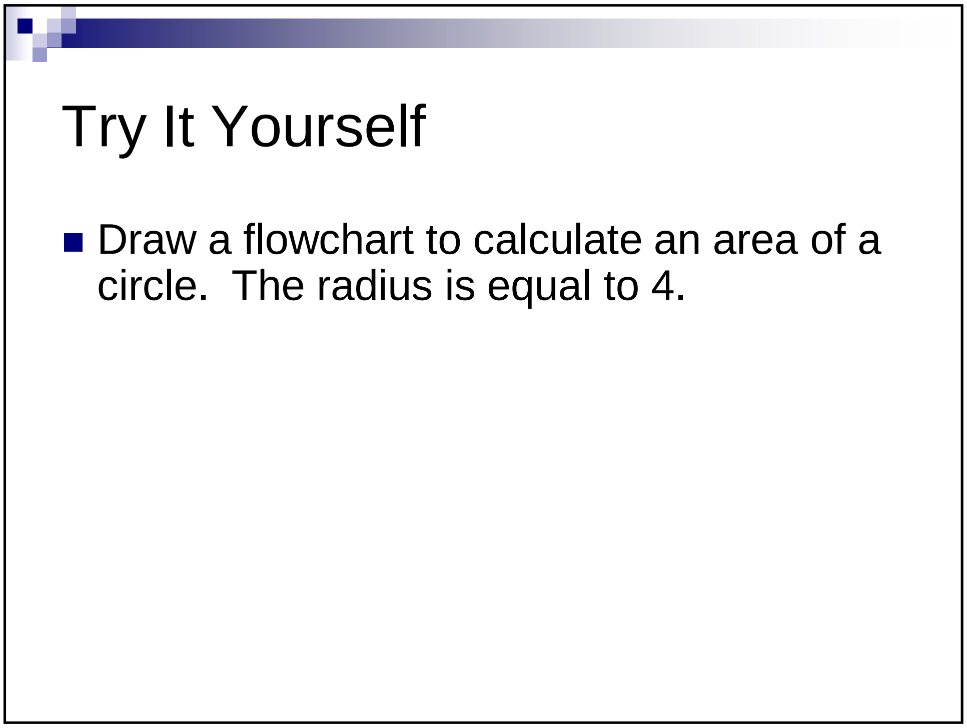
FINISH

```
MOVE.B #10,D0
CLR.B   D1
LOOP…BNE LOOP, CMP.B #$0,D0
ADD.W  (A0),D1
ADDA.L #$2,A0
SUB.B   #$1,D0
END
```



Complete Program

```
START     ORG       $1000

          MOVE.W    #$11,$1100
          MOVE.W    #$22,$1102
          MOVE.W    #$33,$1104
          MOVE.W    #$44,$1106
          MOVE.W    #$55,$1108
          MOVE.W    #$66,$110A
          MOVE.W    #$77,$110C
          MOVE.W    #$88,$110E
          MOVE.W    #$99,$1110
          MOVE.W    #$11,$1112

          MOVEA.L   #$1100,A0
          MOVE.B    #10,D0
          CLR.B     D1

LOOP      ADD.W     (A0),D1
          ADDA.L    #$2,A0
          SUB.B     #$1,D0
          CMP.B     #$0,D0
          BNE       LOOP

          END       START
```



Try It Yourself

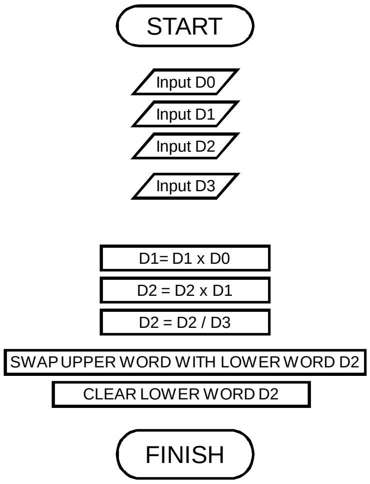
- Draw a flowchart to calculate an area of a circle. The radius is equal to 4.

Complete Program

```
START   ORG      $1000

        MOVE.W   #4, D0
        MOVE.W   #4, D1
        MOVE.W   #22, D2
        MOVE.W   #7, D3

        MULU     D0, D1
        MULU     D1, D2
        DIVU.W   D3, D2
        SWAP     D2
        CLR.W    D2

        END      START
```

Complete Program

```
START   ORG      $1000

        MOVE.W   #4, D0
        MOVE.W   #4, D1
        MOVE.W   #22, D2
        MOVE.W   #7, D3

        MULU     D0, D1
        MULU     D1, D2
        DIVU.W   D3, D2
        SWAP     D2
        CLR.W    D2

        END      START
```



START

Input D0

Input D1

Input D2

Input D3

D1= D1 x D0

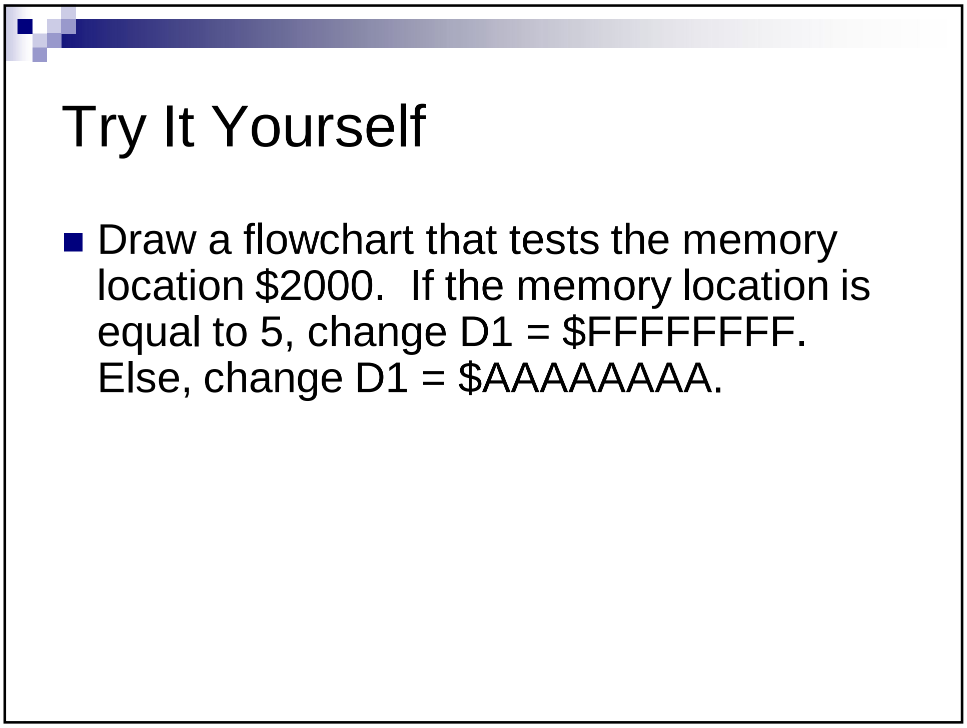
D2 = D2 x D1

D2 = D2 / D3

SWAP UPPER WORD WITH LOWER WORD D2

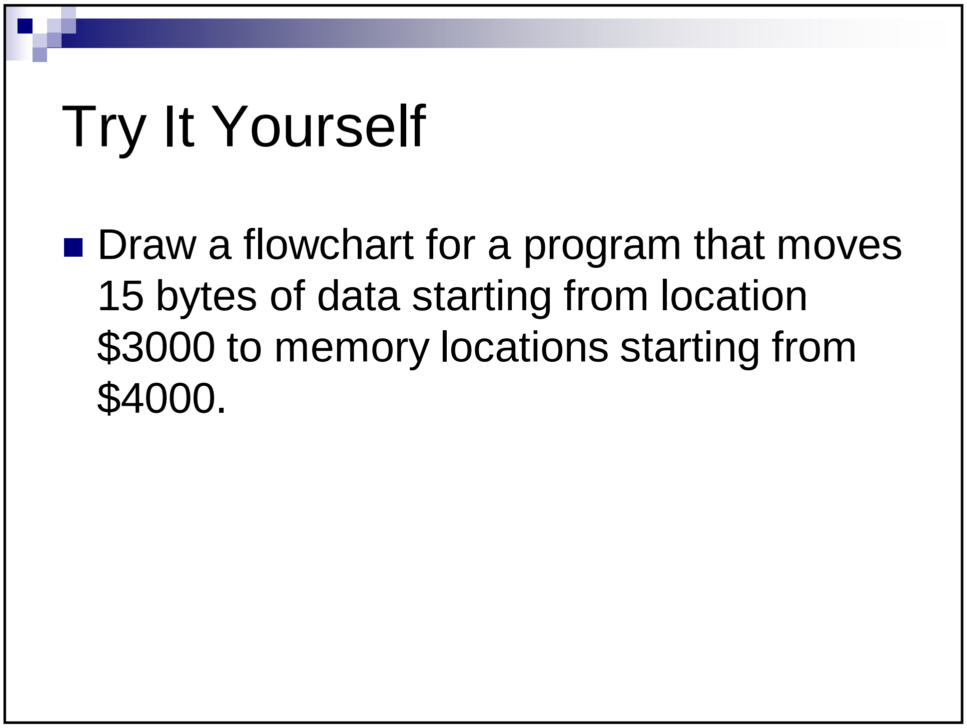
CLEAR LOWER WORD D2

FINISH



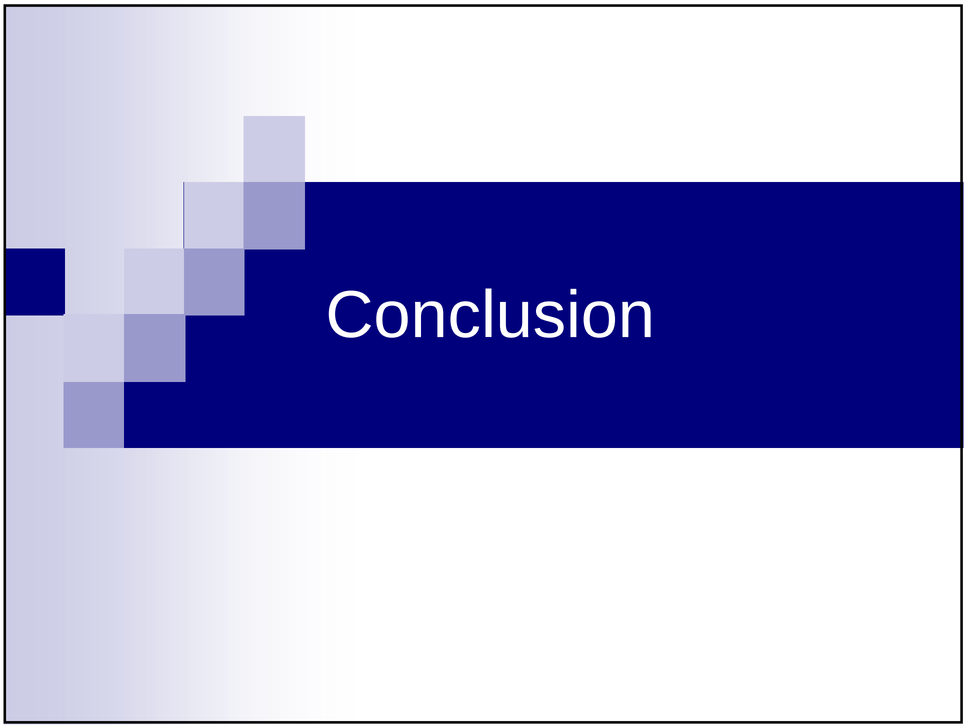
Try It Yourself

- Draw a flowchart that tests the memory location $2000. If the memory location is equal to 5, change D1 = $FFFFFFFF. Else, change D1 = $AAAAAAAA.

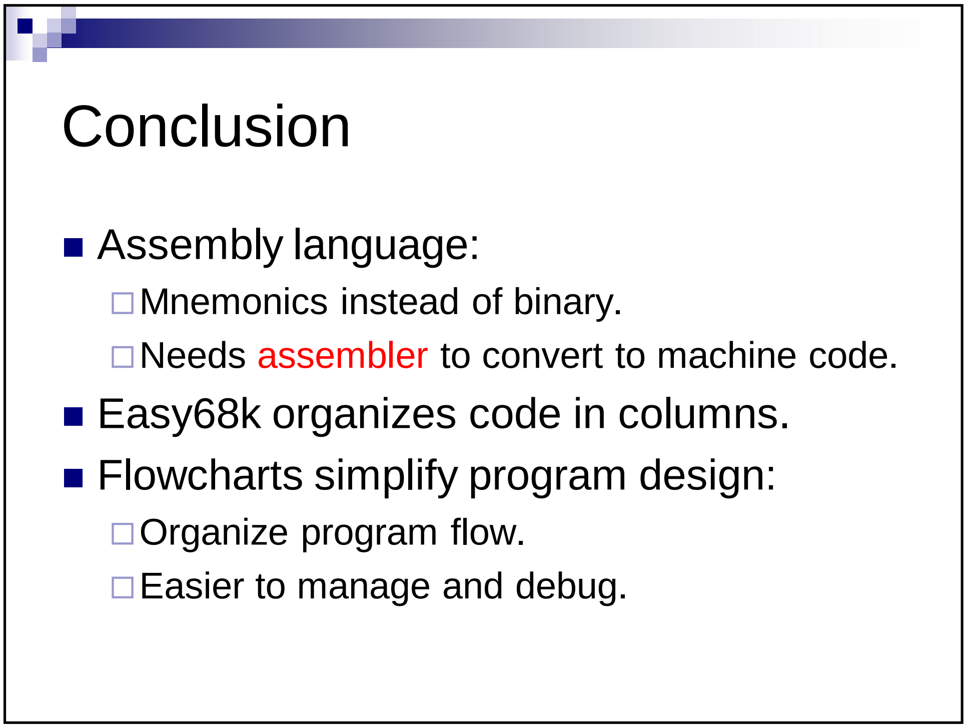


Try It Yourself

- Draw a flowchart for a program that moves 15 bytes of data starting from location $3000 to memory locations starting from $4000.

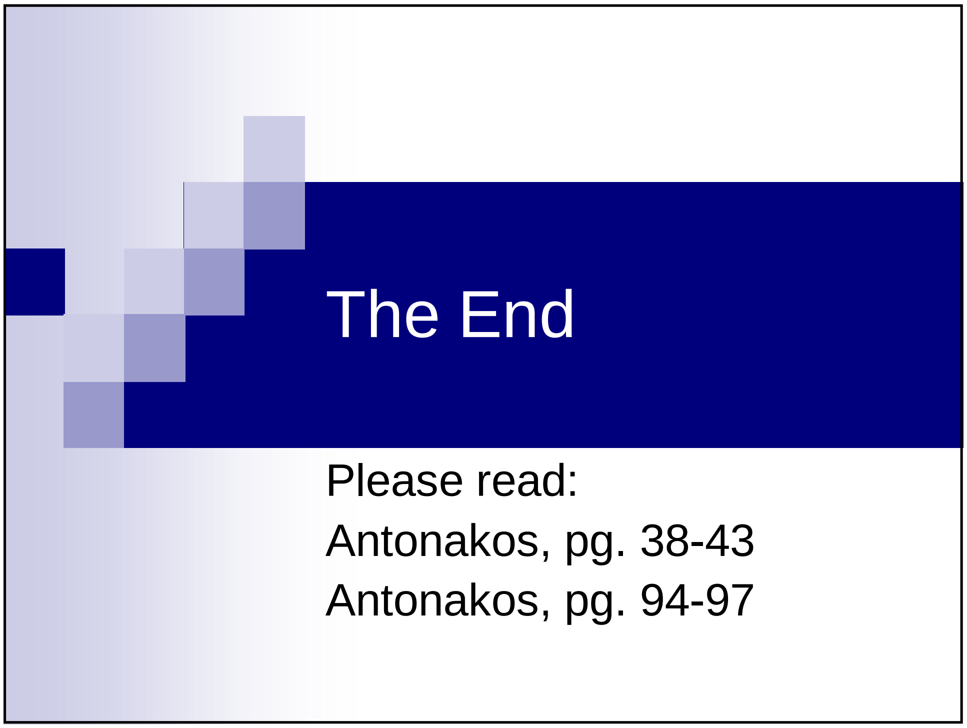


Conclusion



Conclusion

- Assembly language:
  - □Mnemonics instead of binary.
  - □Needs assembler to convert to machine code.
- Easy68k organizes code in columns.
- Flowcharts simplify program design:
  - □Organize program flow.
  - □Easier to manage and debug.



The End

Please read:

Antonakos, pg. 38-43

Antonakos, pg. 94-97