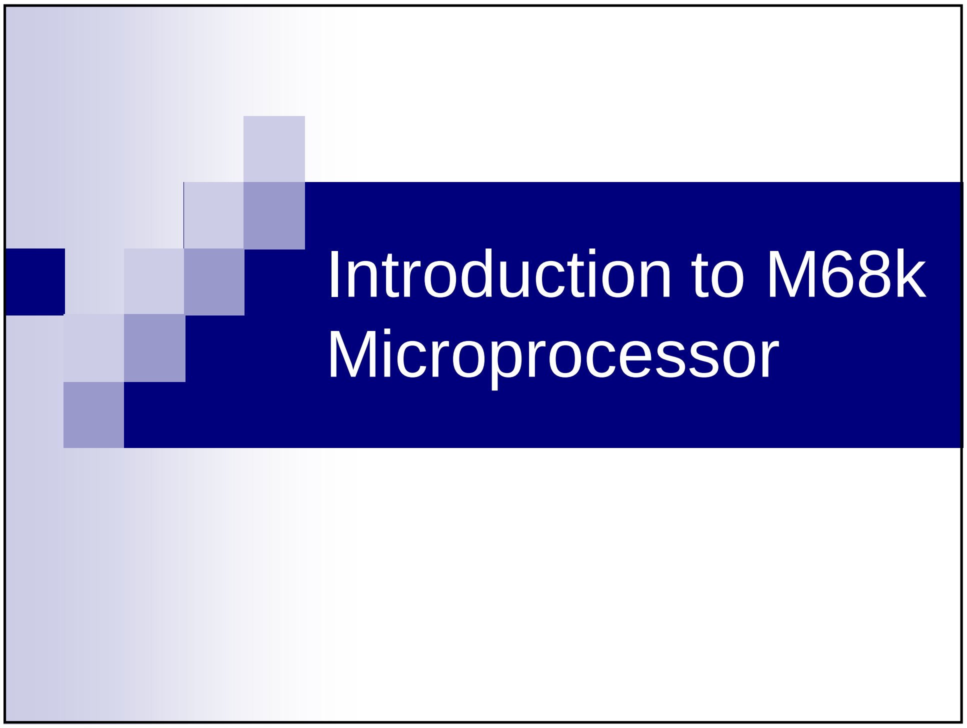
68k Software Model

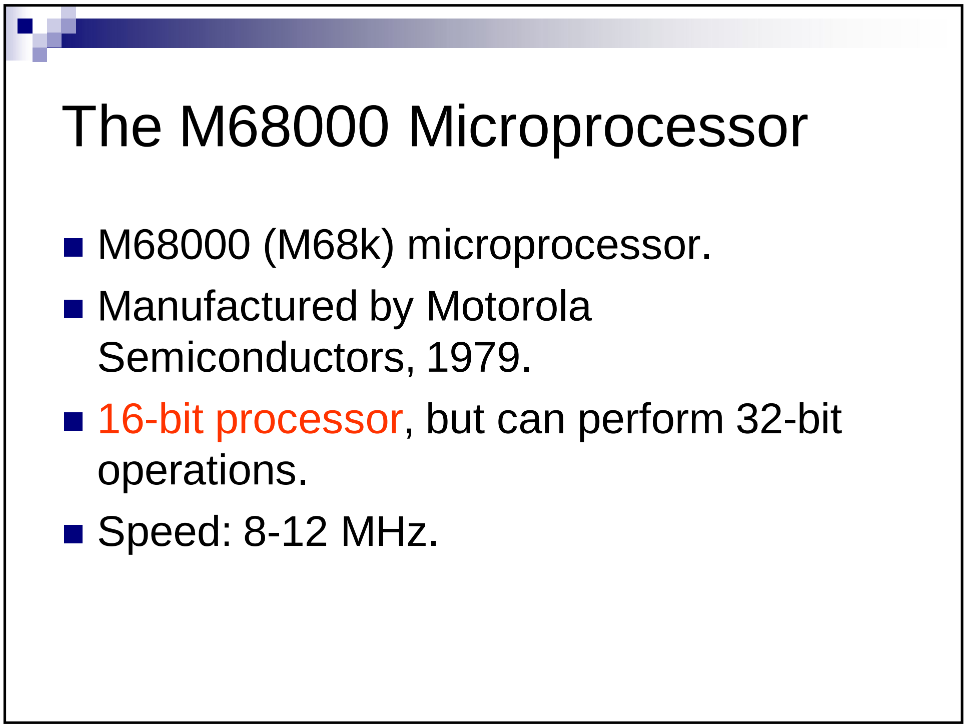
ECE511: Digital System & Microprocessor

What we are going to learn in this session:

- Introduction to M68k microprocessor.
- Software model of M68k:
  - □ M68k internal architecture:
    - Registers in M68k.
    - Their functions.
  - □ Programmer-side view:
    - What you will use as a programmer.



Introduction to M68k Microprocessor



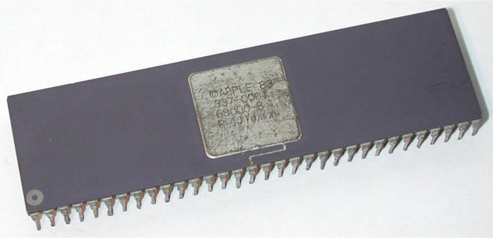
The M68000 Microprocessor

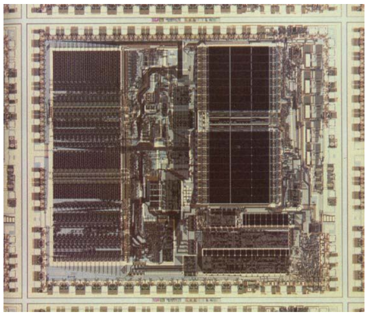
- M68000 (M68k) microprocessor.
- Manufactured by Motorola Semiconductors, 1979.
- 16-bit processor, but can perform 32-bit operations.
- Speed: 8-12 MHz.

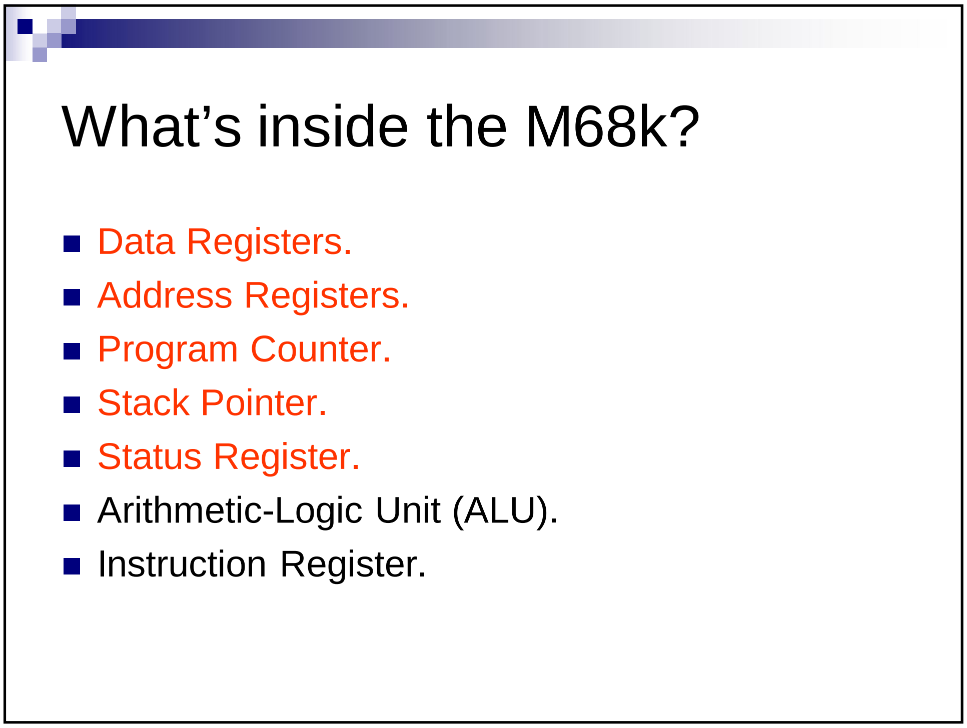
The M68k Microprocessor

- Very advanced compared to 8-bit processors:
  - 16-bit data bus, 24-bit address bus.
  - Can execute instructions twice as fast.
- Still available today:
  - Simple, practical commands.
  - Robust: can be used for many applications.

The M68k Microprocessor

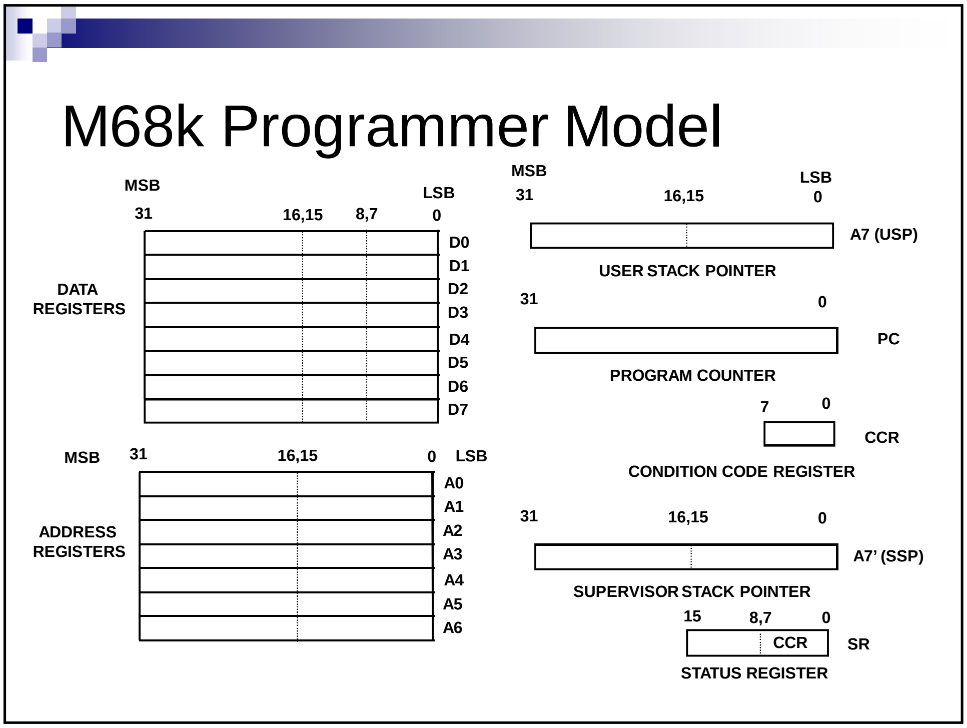






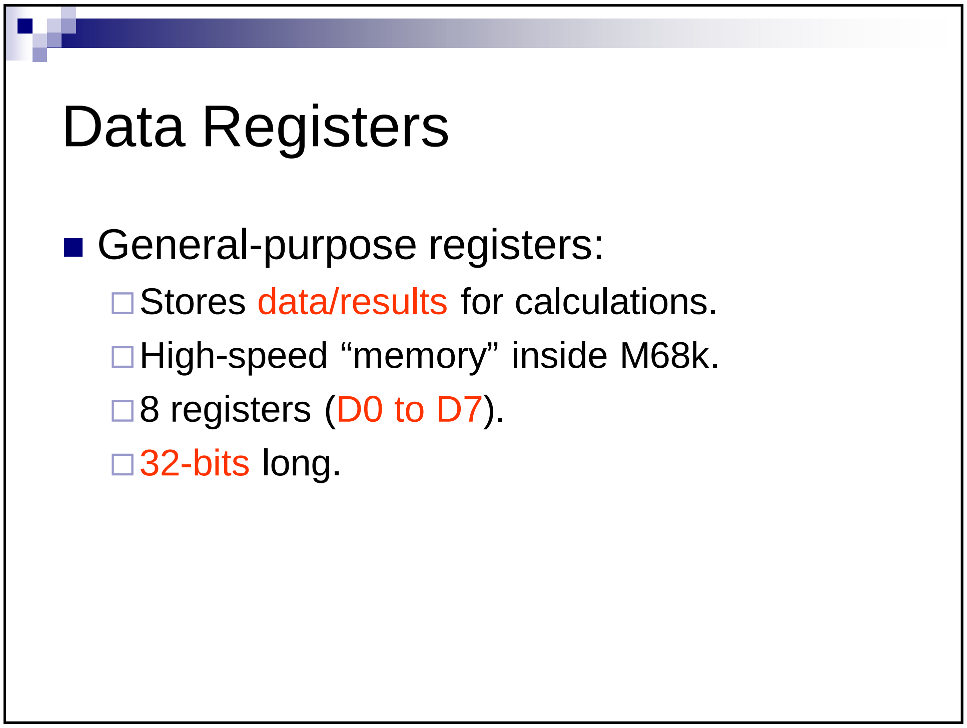
What's inside the M68k?

- Data Registers.
- Address Registers.
- Program Counter.
- Stack Pointer.
- Status Register.
- Arithmetic-Logic Unit (ALU).
- Instruction Register.



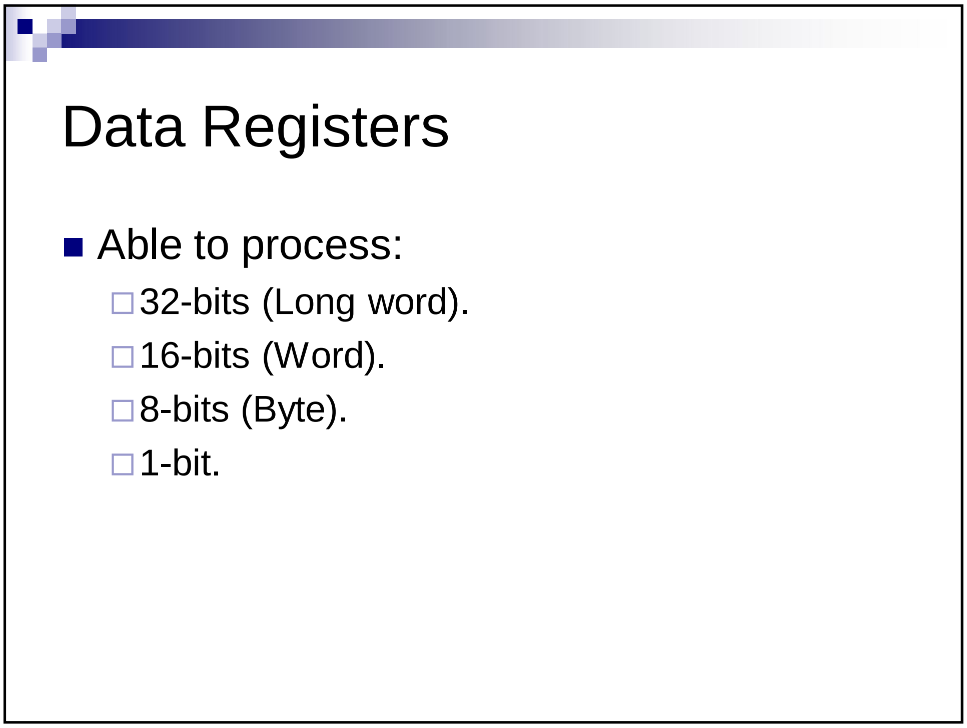
M68k Programmer Model

Data & Address Registers



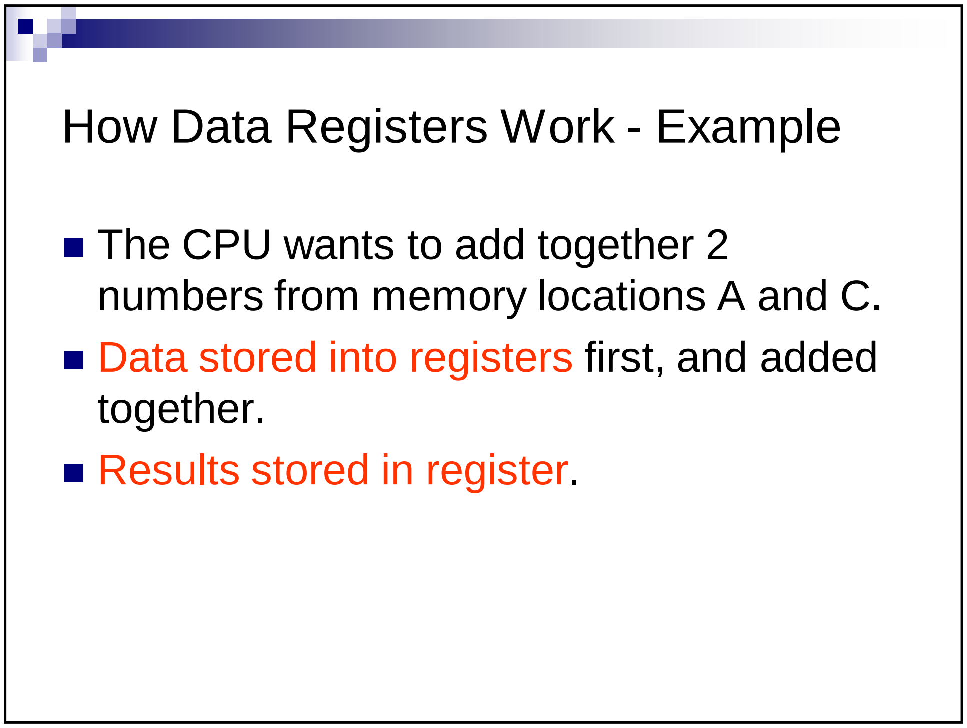
Data Registers

- General-purpose registers:
  - □ Stores data/results for calculations.
  - □ High-speed “memory” inside M68k.
  - □ 8 registers (D0 to D7).
  - □ 32-bits long.



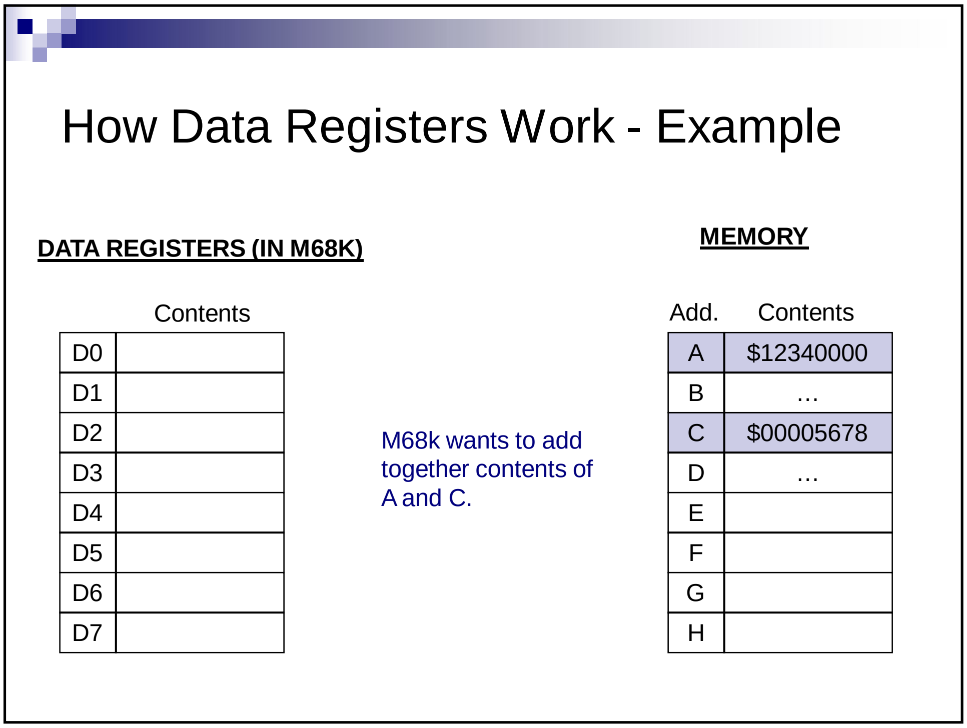
Data Registers

- Able to process:
  - □ 32-bits (Long word).
  - □ 16-bits (Word).
  - □ 8-bits (Byte).
  - □ 1-bit.



How Data Registers Work - Example

- The CPU wants to add together 2 numbers from memory locations A and C.
- Data stored into registers first, and added together.
- Results stored in register.



How Data Registers Work - Example

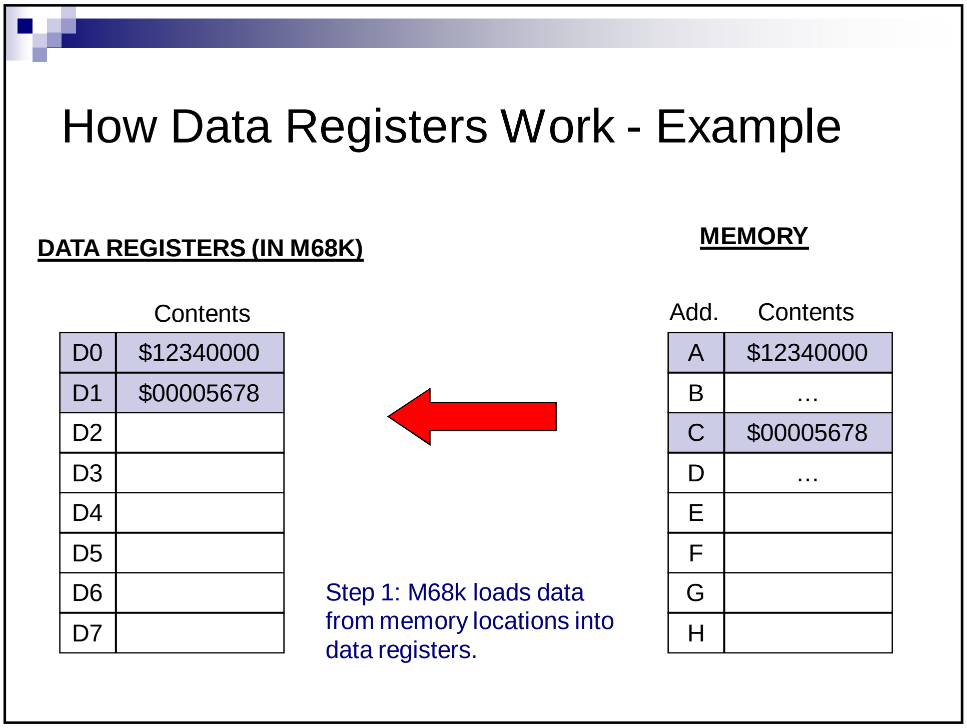
DATA REGISTERS (IN M68K)

	Contents
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

M68k wants to add together contents of A and C.

MEMORY

Add.	Contents
A	$12340000
B	…
C	$00005678
D	…
E	
F	
G	
H	



How Data Registers Work - Example

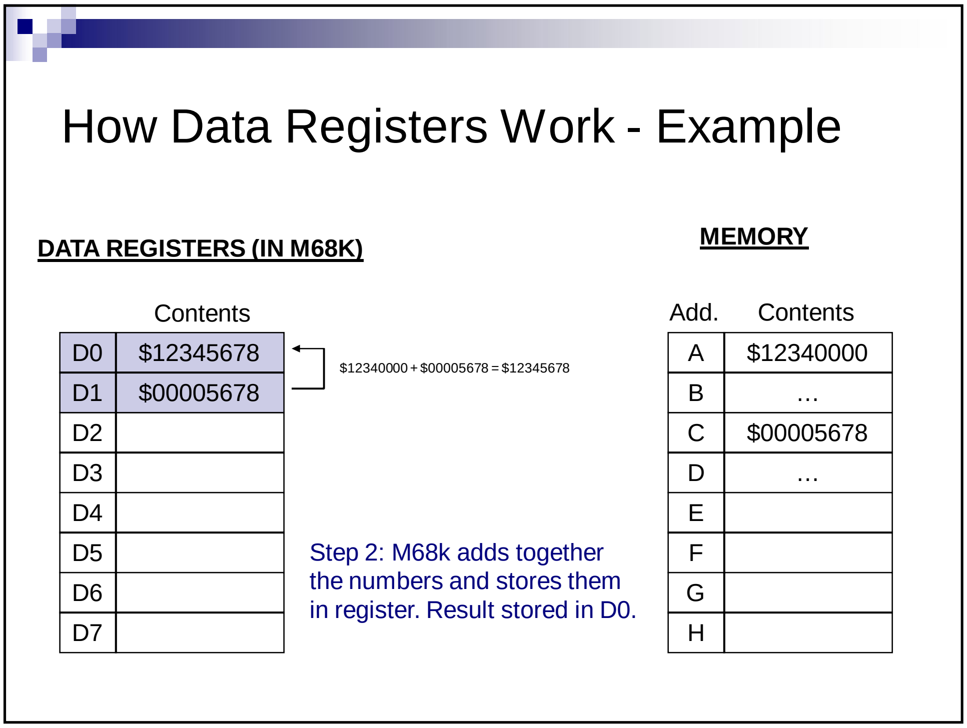
DATA REGISTERS (IN M68K)

	Contents
D0	$12340000
D1	$00005678
D2	
D3	
D4	
D5	
D6	
D7	

Step 1: M68k loads data from memory locations into data registers.

MEMORY

Add.	Contents
A	$12340000
B	…
C	$00005678
D	…
E	
F	
G	
H	



How Data Registers Work - Example

DATA REGISTERS (IN M68K)

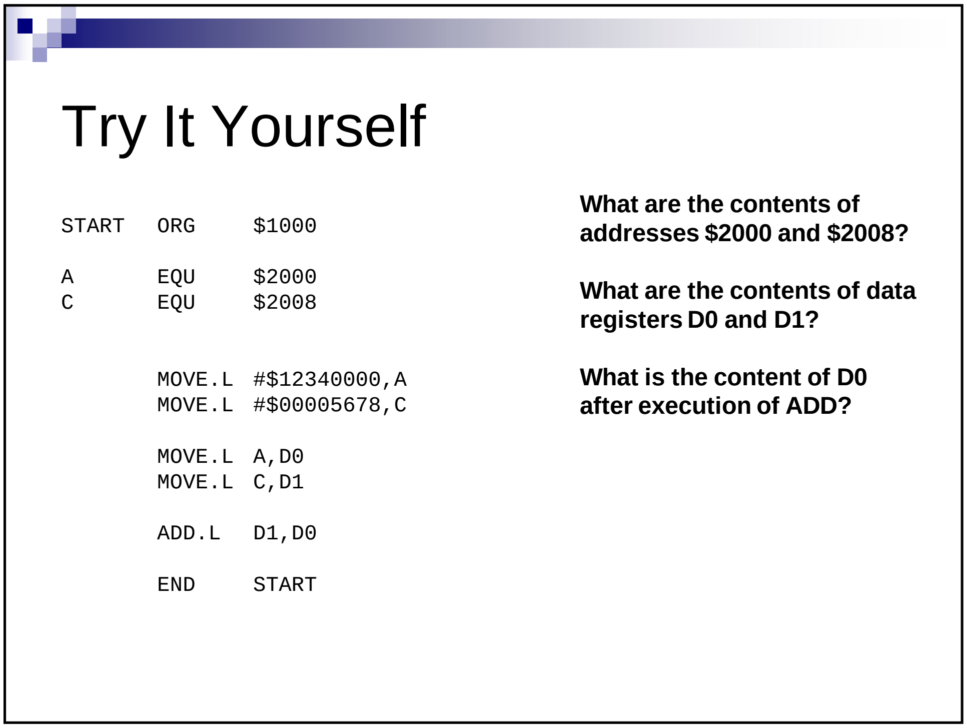
	Contents
D0	$12345678
D1	$00005678
D2	
D3	
D4	
D5	
D6	
D7	

$12340000 + $00005678 = $12345678

Step 2: M68k adds together the numbers and stores them in register. Result stored in D0.

MEMORY

Add.	Contents
A	$12340000
B	…
C	$00005678
D	…
E	
F	
G	
H	



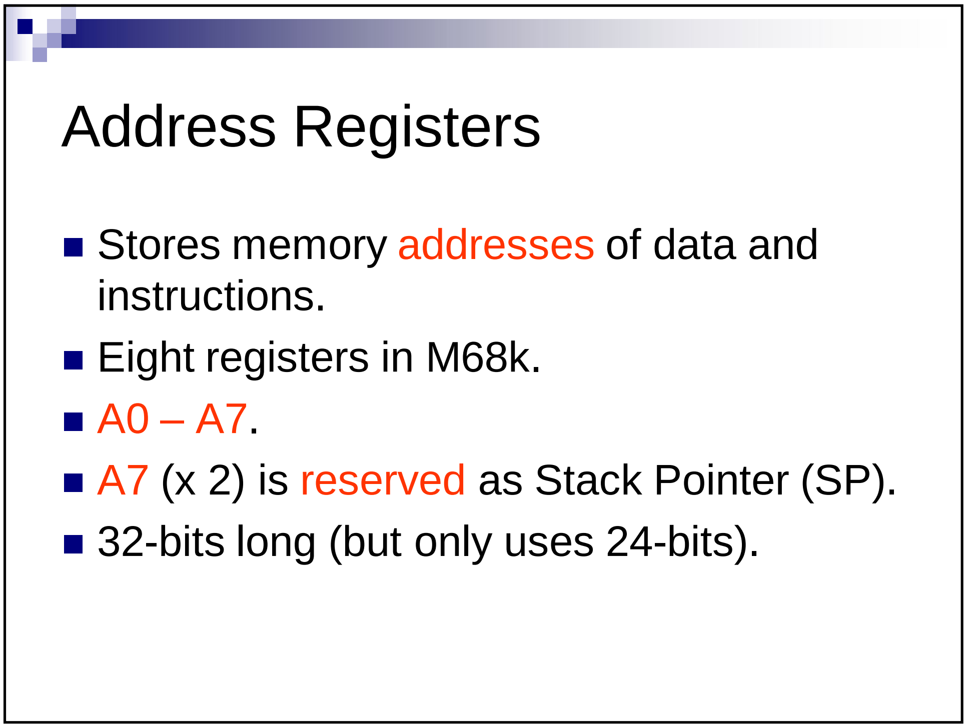
Try It Yourself

```
START   ORG     $1000

A       EQU     $2000
C       EQU     $2008


        MOVE.L  #$12340000,A
        MOVE.L  #$00005678,C

        MOVE.L  A,D0
        MOVE.L  C,D1

        ADD.L   D1,D0

        END     START
```

What are the contents of addresses $2000 and $2008?

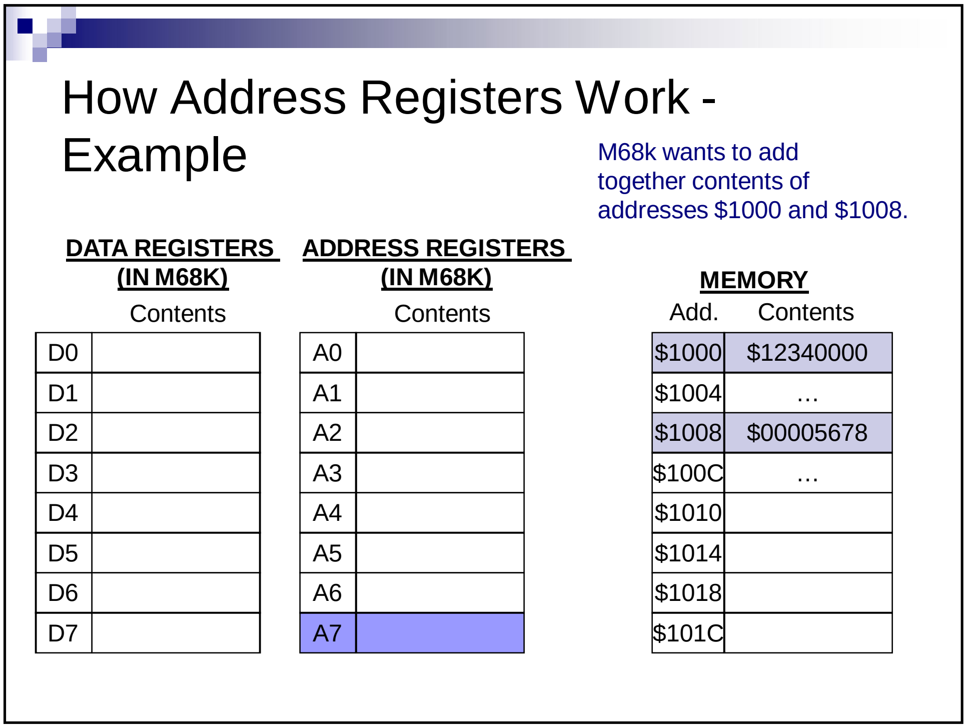
What are the contents of data registers D0 and D1?

What is the content of D0 after execution of ADD?



Address Registers

- Stores memory addresses of data and instructions.
- Eight registers in M68k.
- A0 – A7.
- A7 (x 2) is reserved as Stack Pointer (SP).
- 32-bits long (but only uses 24-bits).



How Address Registers Work - Example

M68k wants to add together contents of addresses $1000 and $1008.

DATA REGISTERS (IN M68K)

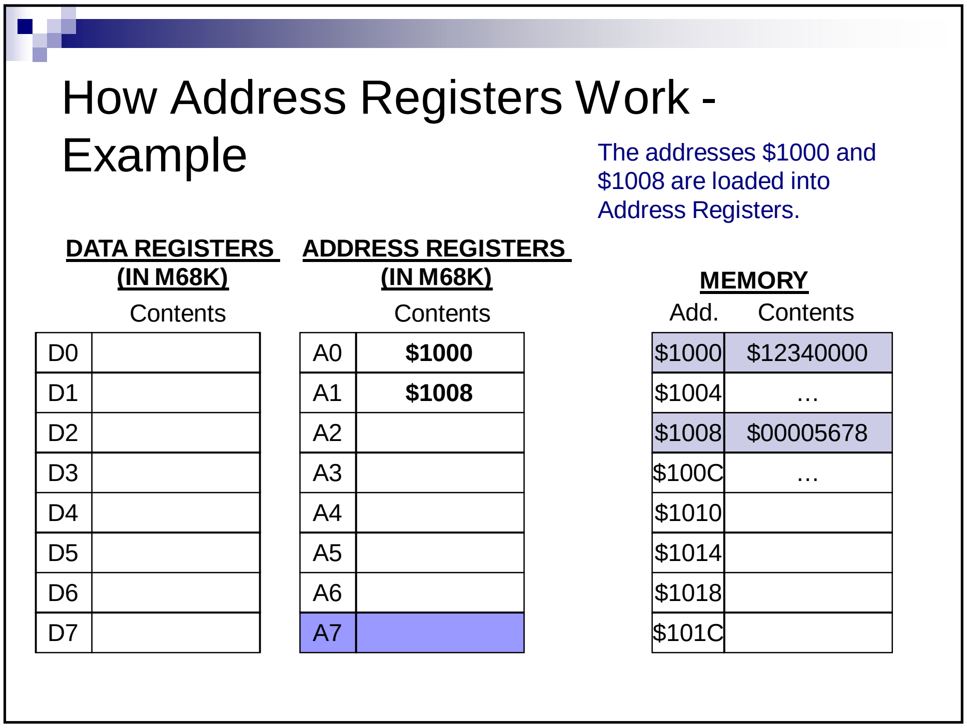
	Contents
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

ADDRESS REGISTERS (IN M68K)

	Contents
A0	
A1	
A2	
A3	
A4	
A5	
A6	
A7	

MEMORY

Add.	Contents
$1000	$12340000
$1004	…
$1008	$00005678
$100C	…
$1010	
$1014	
$1018	
$101C	



How Address Registers Work - Example

The addresses $1000 and $1008 are loaded into Address Registers.

DATA REGISTERS (IN M68K)

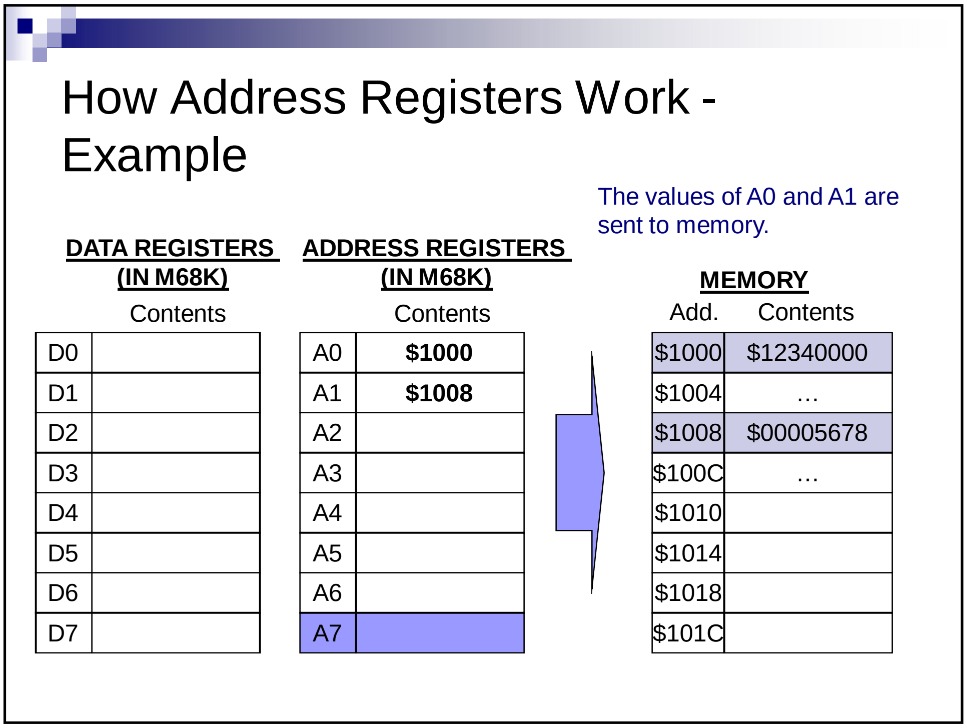
	Contents
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

ADDRESS REGISTERS (IN M68K)

	Contents
A0	**$1000**
A1	**$1008**
A2	
A3	
A4	
A5	
A6	
A7	

MEMORY

Add.	Contents
$1000	$12340000
$1004	…
$1008	$00005678
$100C	…
$1010	
$1014	
$1018	
$101C	



How Address Registers Work - Example

The values of A0 and A1 are sent to memory.

DATA REGISTERS (IN M68K)

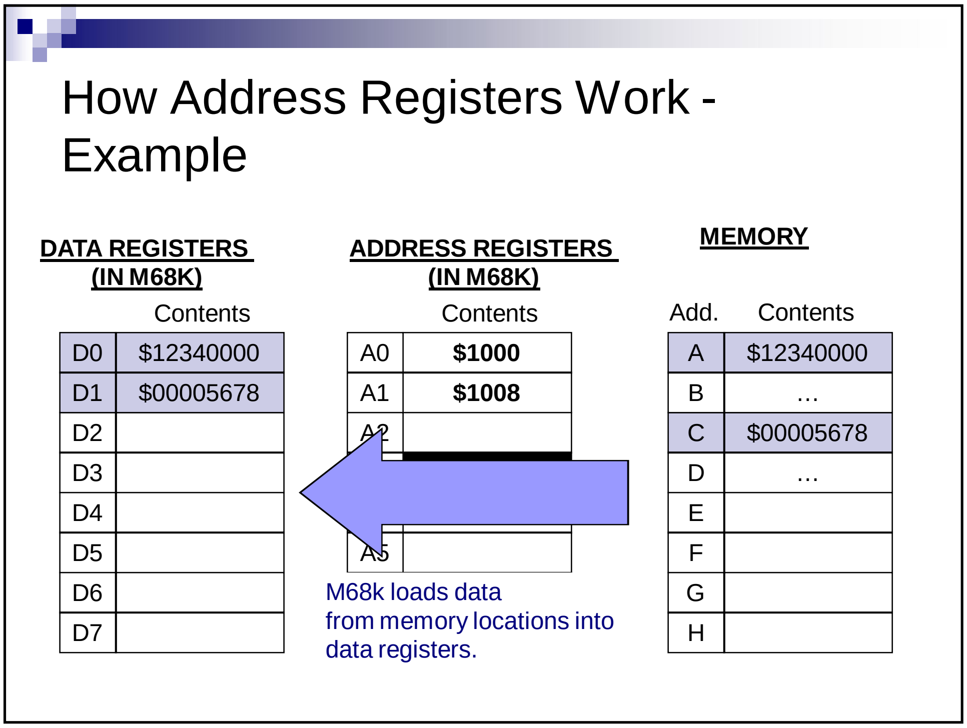
	Contents
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

ADDRESS REGISTERS (IN M68K)

	Contents
A0	**$1000**
A1	**$1008**
A2	
A3	
A4	
A5	
A6	
A7	

MEMORY

Add.	Contents
$1000	$12340000
$1004	…
$1008	$00005678
$100C	…
$1010	
$1014	
$1018	
$101C	



How Address Registers Work - Example

DATA REGISTERS (IN M68K)

	Contents
D0	$12340000
D1	$00005678
D2	
D3	
D4	
D5	
D6	
D7	

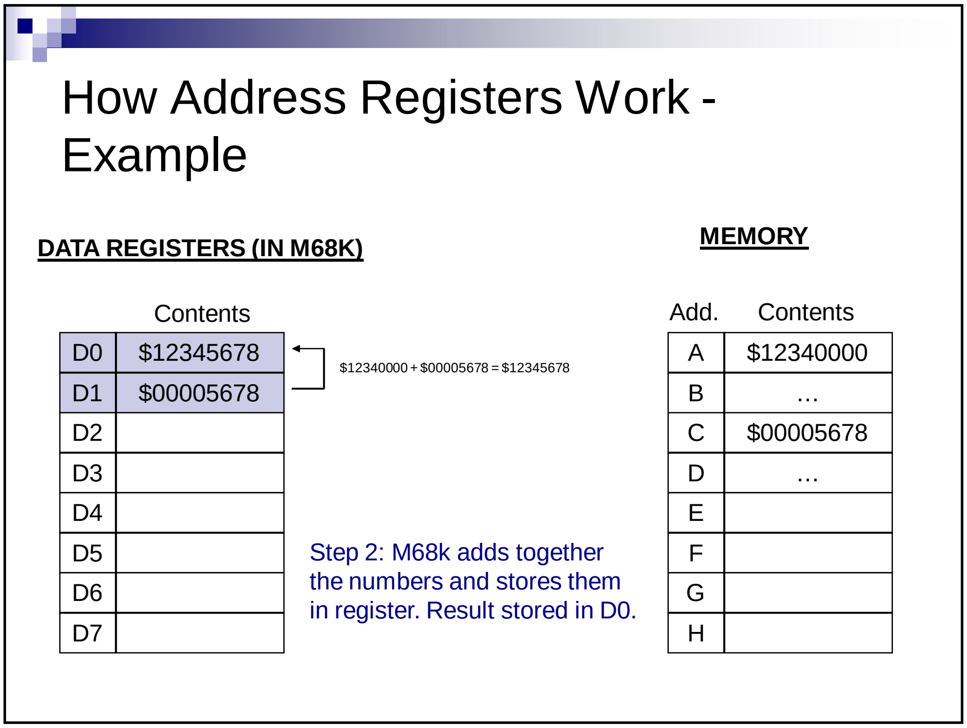
ADDRESS REGISTERS (IN M68K)

	Contents
A0	**$1000**
A1	**$1008**
A2	
A5	

MEMORY

Add.	Contents
A	$12340000
B	…
C	$00005678
D	…
E	
F	
G	
H	

M68k loads data from memory locations into data registers.



How Address Registers Work - Example

DATA REGISTERS (IN M68K)

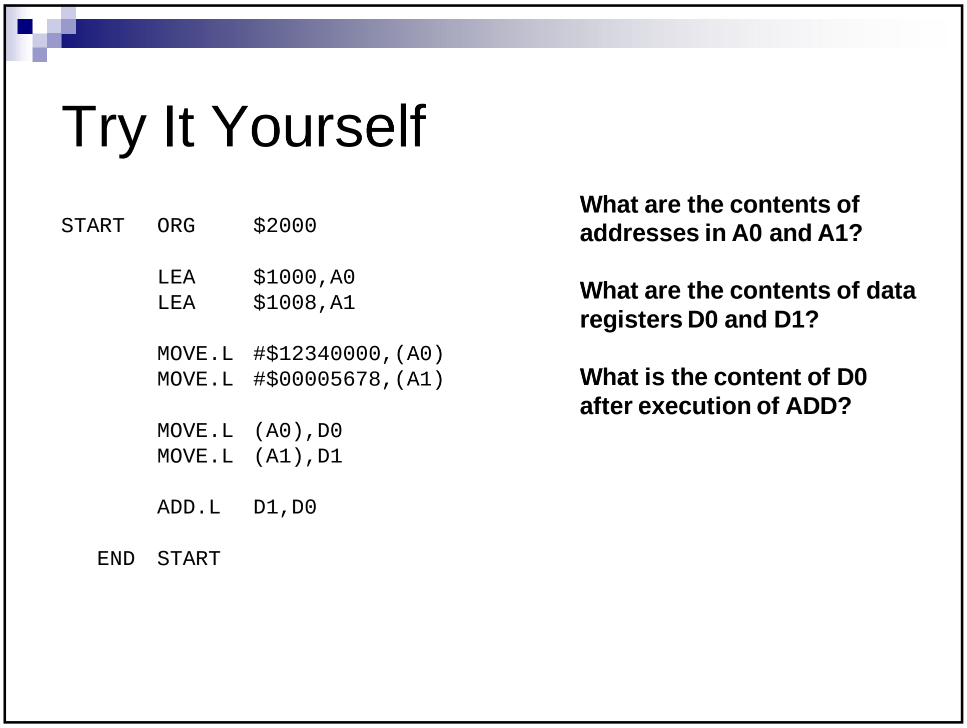
	Contents
D0	$12345678
D1	$00005678
D2	
D3	
D4	
D5	
D6	
D7	

$12340000 + $00005678 = $12345678

MEMORY

Add.	Contents
A	$12340000
B	…
C	$00005678
D	…
E	
F	
G	
H	

Step 2: M68k adds together the numbers and stores them in register. Result stored in D0.



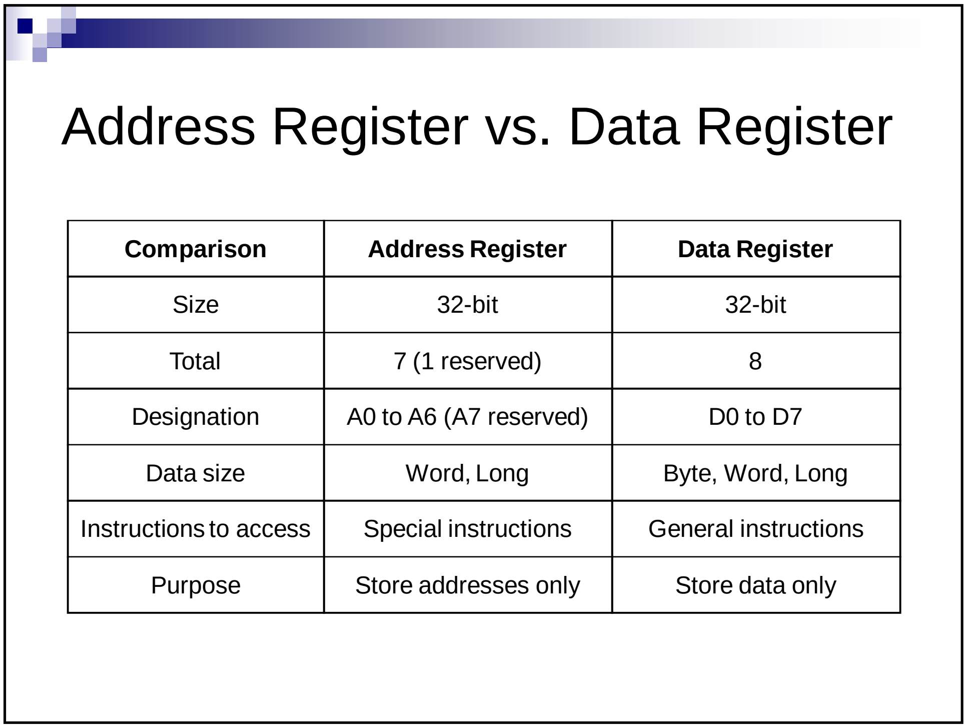
Try It Yourself

```
START   ORG     $2000

        LEA     $1000,A0
        LEA     $1008,A1

        MOVE.L  #$12340000,(A0)
        MOVE.L  #$00005678,(A1)

        MOVE.L  (A0),D0
        MOVE.L  (A1),D1

        ADD.L   D1,D0

   END  START
```

What are the contents of addresses in A0 and A1?

What are the contents of data registers D0 and D1?

What is the content of D0 after execution of ADD?



Address Register vs. Data Register

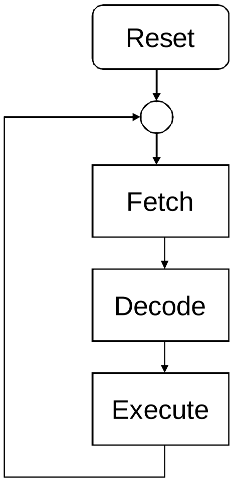
Comparison	**Address Register**	**Data Register**
Size	32-bit	32-bit
Total	7 (1 reserved)	8
Designation	A0 to A6 (A7 reserved)	D0 to D7
Data size	Word, Long	Byte, Word, Long
Instructions to access	Special instructions	General instructions
Purpose	Store addresses only	Store data only

Program Counter

Program Counter (PC)

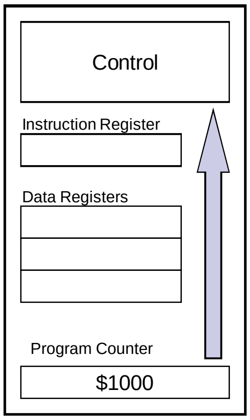
- Instructions must occur in correct sequence for program to run properly.
- PC makes sure this happens:
  - □Special-purpose register inside CPU.
  - □Keeps track of address of next instruction.
  - □32-bits: can point to any location in memory.

Remember this cycle?



Fetch – Step 1

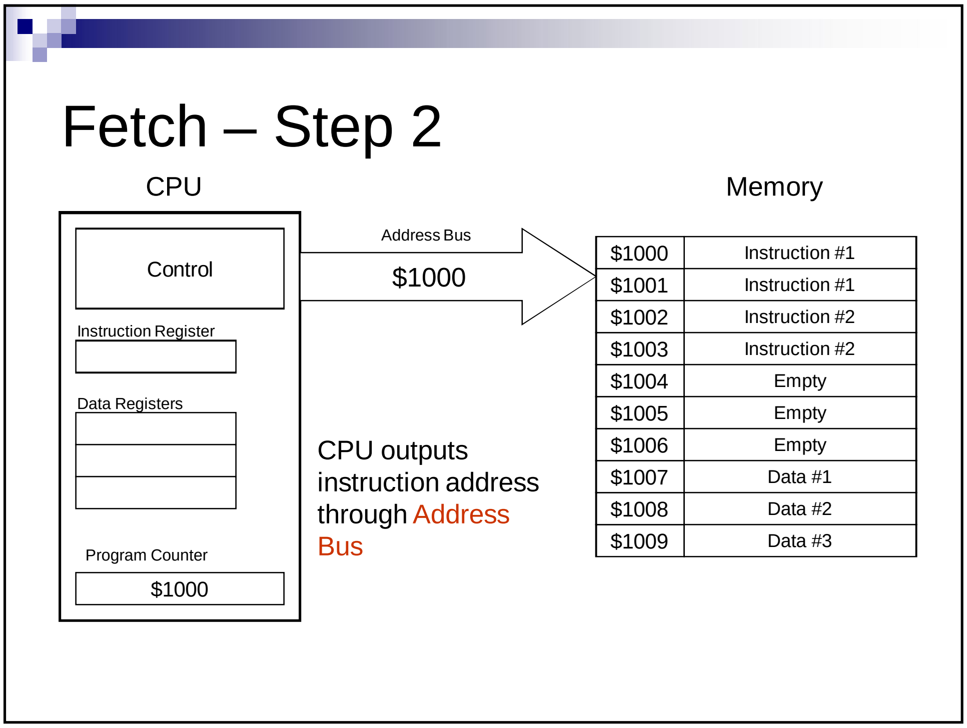
CPU



CPU gets instruction address from PC

Memory

$1000	Instruction #1
$1001	Instruction #1
$1002	Instruction #2
$1003	Instruction #2
$1004	Empty
$1005	Empty
$1006	Empty
$1007	Data #1
$1008	Data #2
$1009	Data #3



Fetch – Step 2

CPU

Control

Instruction Register

Data Registers

Program Counter

$1000

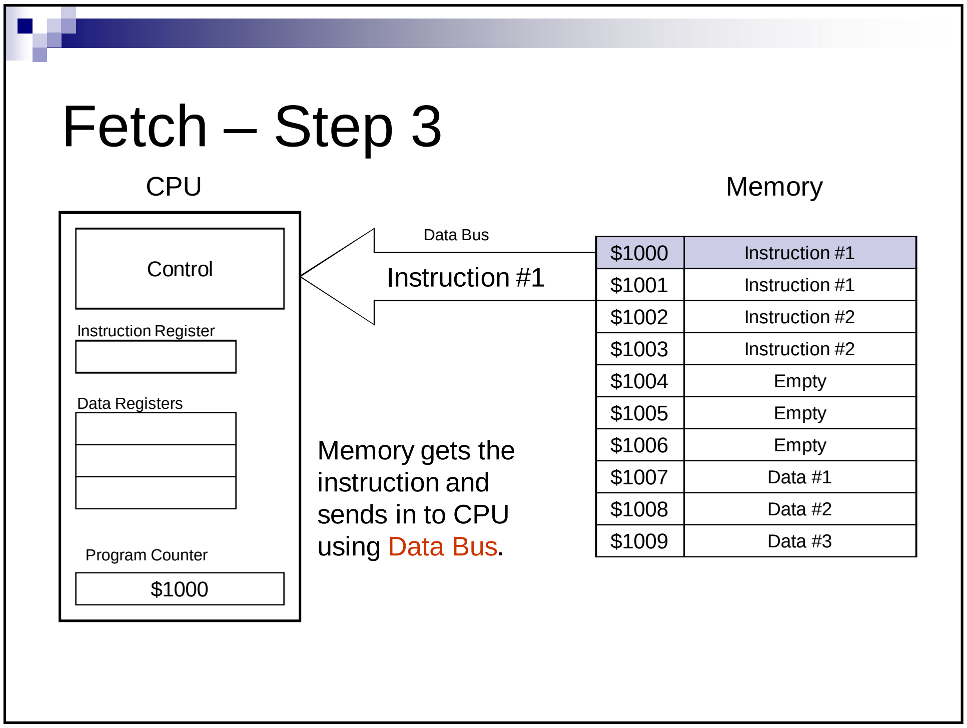
Address Bus

$1000

Memory

$1000	Instruction #1
$1001	Instruction #1
$1002	Instruction #2
$1003	Instruction #2
$1004	Empty
$1005	Empty
$1006	Empty
$1007	Data #1
$1008	Data #2
$1009	Data #3

CPU outputs instruction address through Address Bus



Fetch – Step 3

CPU

Control

Instruction Register

Data Registers

Program Counter

$1000

Data Bus

Instruction #1

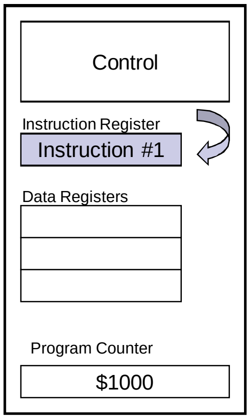
Memory

$1000	Instruction #1
$1001	Instruction #1
$1002	Instruction #2
$1003	Instruction #2
$1004	Empty
$1005	Empty
$1006	Empty
$1007	Data #1
$1008	Data #2
$1009	Data #3

Memory gets the instruction and sends in to CPU using Data Bus.

Fetch – Step 4

CPU



Control

Instruction Register

Instruction #1

Data Registers

Program Counter

$1000

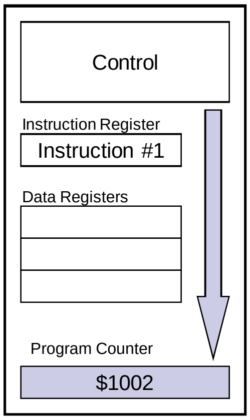
CPU stores instruction in Instruction Register

Memory

$1000	Instruction #1
$1001	Instruction #1
$1002	Instruction #2
$1003	Instruction #2
$1004	Empty
$1005	Empty
$1006	Empty
$1007	Data #1
$1008	Data #2
$1009	Data #3

Fetch – Step 5

CPU



Control

Instruction Register

Instruction #1

Data Registers

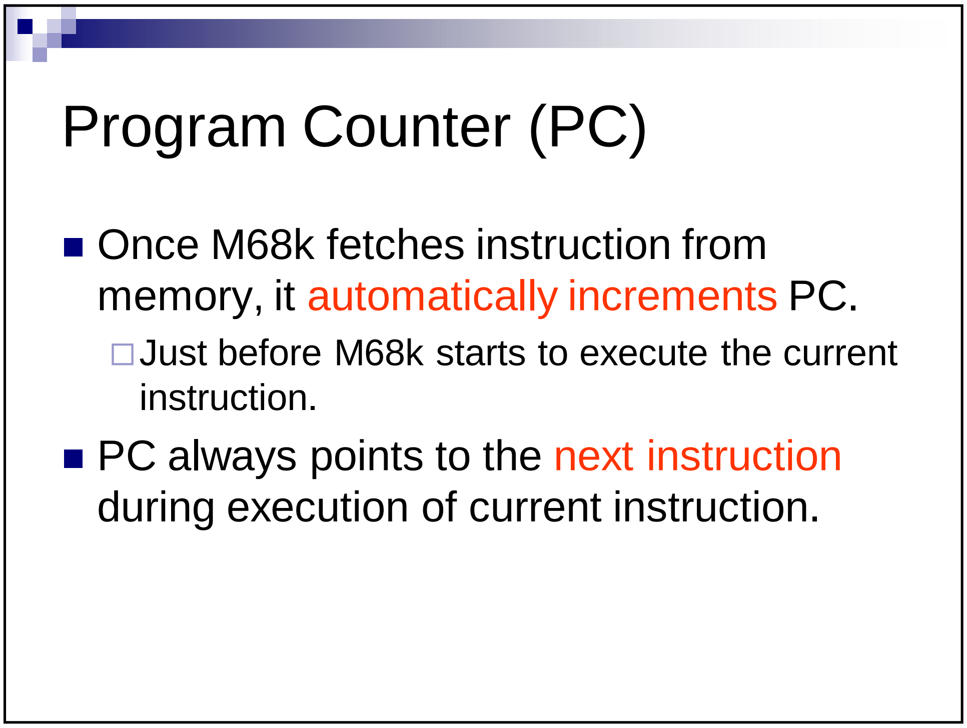
Program Counter

$1002

After instruction has been loaded, CPU updates Program Counter.

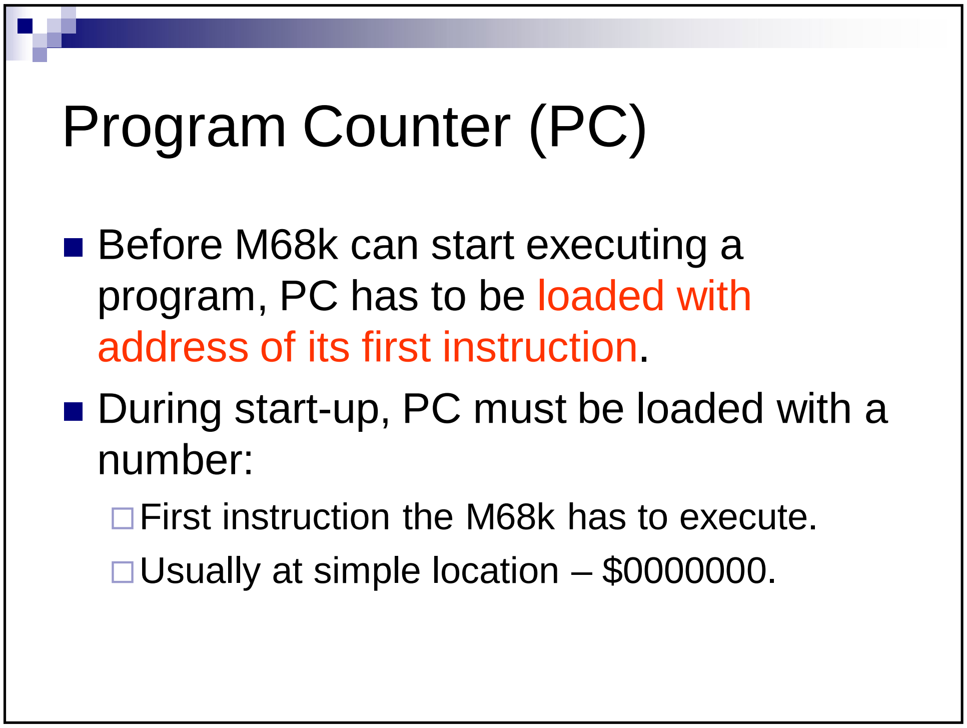
Memory

$1000	Instruction #1
$1001	Instruction #1
$1002	Instruction #2
$1003	Instruction #2
$1004	Empty
$1005	Empty
$1006	Empty
$1007	Data #1
$1008	Data #2
$1009	Data #3



Program Counter (PC)

- Once M68k fetches instruction from memory, it automatically increments PC.
  - Just before M68k starts to execute the current instruction.
- PC always points to the next instruction during execution of current instruction.



Program Counter (PC)

- Before M68k can start executing a program, PC has to be loaded with address of its first instruction.
- During start-up, PC must be loaded with a number:
  - First instruction the M68k has to execute.
  - Usually at simple location – $0000000.

Program Counter (PC)

- PC can be loaded with new value to “jump” over instructions.
- When the PC is modified, new instruction is taken from new PC value, not the address in sequence.
- Can use this technique in subroutines, exceptions and branches.

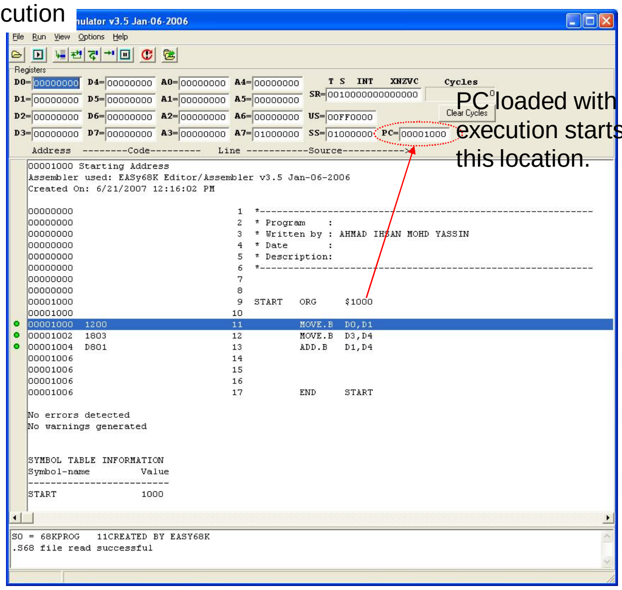
Try It Yourself

```
START   ORG     $1000

        MOVE.B  D0,D1
        MOVE.B  D3,D4
        ADD.B   D1,D4


        END     START
```

What are the contents of memory address starting from $1000?

Execute the program step by step. Notice that the value of PC changes automatically after each instruction.

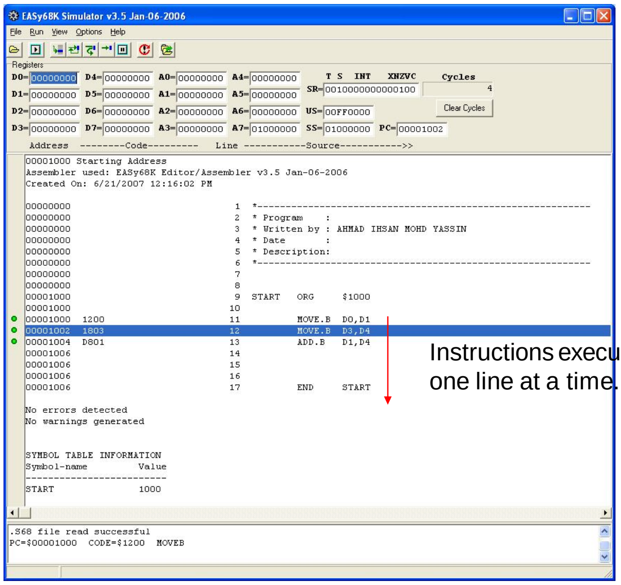
* Normal execution



```
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: 6/21/2007 12:16:02 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00000000                            7
00000000                            8
00001000                            9  START    ORG     $1000
00001000                           10
00001000  1200                     11          MOVE.B  D0,D1
00001002  1803                     12          MOVE.B  D3,D4
00001004  D801                     13          ADD.B   D1,D4
00001006                           14
00001006                           15
00001006                           16
00001006                           17          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
```

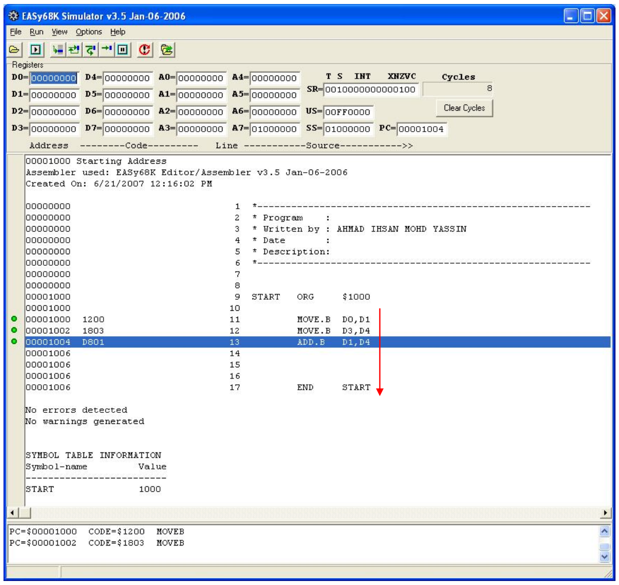
PC loaded with $1000, execution starts with this location.


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=0010000000000100       4
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001002
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 12:16:02 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00000000                            7
00000000                            8
00001000                            9  START    ORG     $1000
00001000                           10
00001000  1200                     11          MOVE.B  D0,D1
00001002  1803                     12          MOVE.B  D3,D4
00001004  D801                     13          ADD.B   D1,D4
00001006                           14
00001006                           15
00001006                           16
00001006                           17          END     START

No errors detected
No warnings generated


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

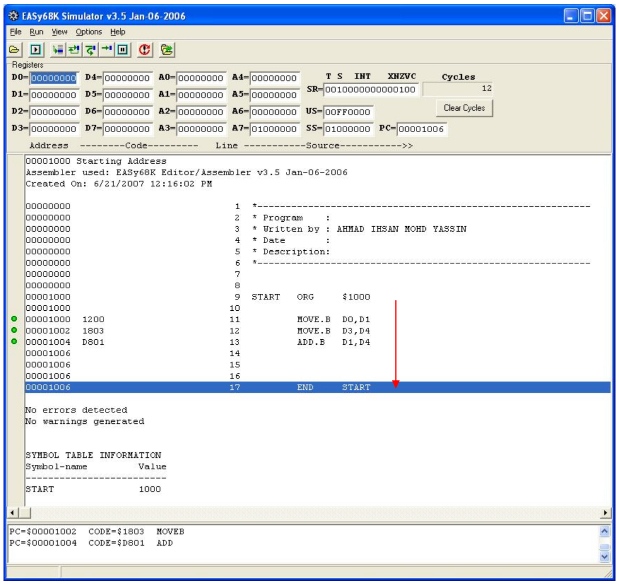
```
.S68 file read successful
PC=$00001000  CODE=$1200  MOVEB
```

Instructions executed one line at a time.



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

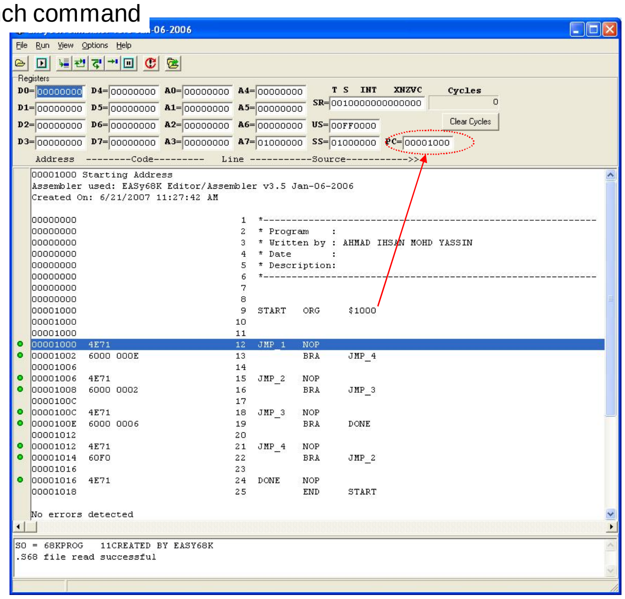
```
D0=00000000  D4=00000000  A0=00000000  A4=00000000     T S  INT   XNZVC     Cycles
D1=00000000  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000100           8
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001004
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 12:16:02 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00000000                            7
00000000                            8
00001000                            9  START  ORG    $1000
00001000                           10
00001000  1200                     11         MOVE.B  D0,D1
00001002  1803                     12         MOVE.B  D3,D4
00001004  D801                     13         ADD.B   D1,D4
00001006                           14
00001006                           15
00001006                           16
00001006                           17         END     START

No errors detected
No warnings generated


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

PC=$00001000   CODE=$1200   MOVEB
PC=$00001002   CODE=$1803   MOVEB
```



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

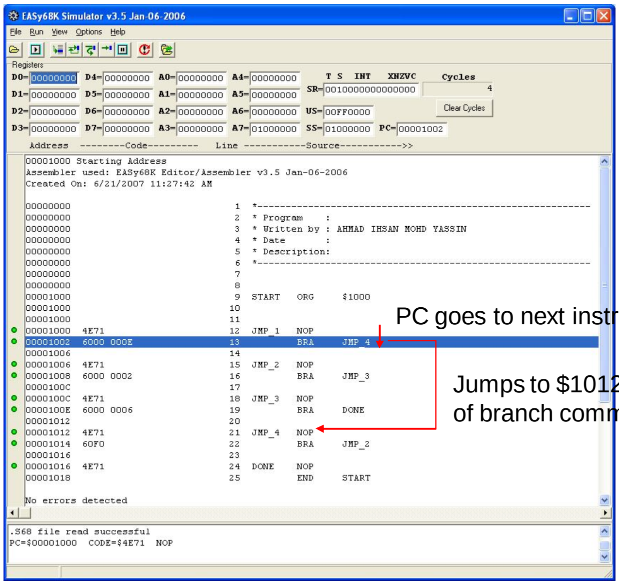
```
D0=00000000  D4=00000000  A0=00000000  A4=00000000     T S  INT   XNZVC     Cycles
D1=00000000  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000100          12
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001006
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 12:16:02 PM

00000000                            1  *-----------------------------------------------------------
00000000                            2  * Program    :
00000000                            3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                            4  * Date       :
00000000                            5  * Description:
00000000                            6  *-----------------------------------------------------------
00000000                            7
00000000                            8
00001000                            9  START  ORG    $1000
00001000                           10
00001000  1200                     11         MOVE.B  D0,D1
00001002  1803                     12         MOVE.B  D3,D4
00001004  D801                     13         ADD.B   D1,D4
00001006                           14
00001006                           15
00001006                           16
00001006                           17         END     START

No errors detected
No warnings generated


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

PC=$00001002   CODE=$1803   MOVEB
PC=$00001004   CODE=$D801   ADD
```

* Using Branch command



```
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: 6/21/2007 11:27:42 AM

00000000                          1  *------------------------------------------------------------
00000000                          2  * Program    :
00000000                          3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                          4  * Date       :
00000000                          5  * Description:
00000000                          6  *------------------------------------------------------------
00000000                          7
00000000                          8
00001000                          9  START   ORG     $1000
00001000                         10
00001000                         11
00001000  4E71                   12  JMP_1   NOP
00001002  6000 000E              13          BRA     JMP_4
00001006                         14
00001006  4E71                   15  JMP_2   NOP
00001008  6000 0002              16          BRA     JMP_3
0000100C                         17
0000100C  4E71                   18  JMP_3   NOP
0000100E  6000 0006              19          BRA     DONE
00001012                         20
00001012  4E71                   21  JMP_4   NOP
00001014  60F0                   22          BRA     JMP_2
00001016                         23
00001016  4E71                   24  DONE    NOP
00001018                         25          END     START

No errors detected
```

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



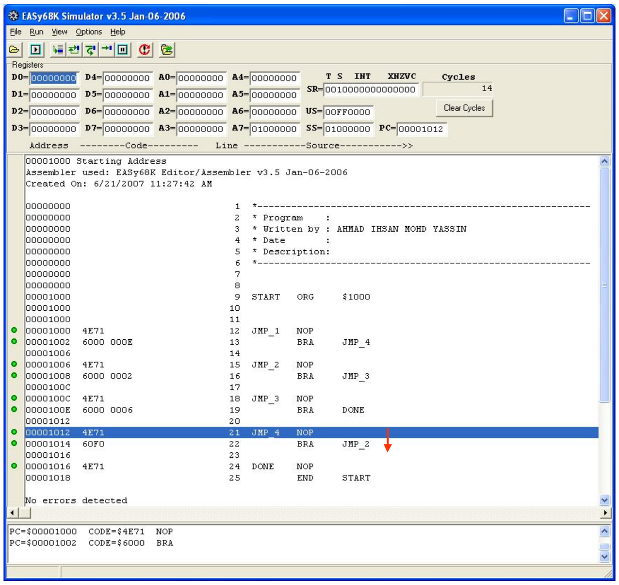
EASy68K Simulator v3.5 Jan-06-2006
File Run View Options Help
Registers
Clear Cycles

```
D0=00000000  D4=00000000  A0=00000000  A4=00000000      T S  INT   XNZVC      Cycles
D1=00000000  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000000            4
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001002
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 11:27:42 AM

00000000                          1  *------------------------------------------------------------
00000000                          2  * Program    :
00000000                          3  * Written by : AHMAD IHSAN MOHD YASSIN
00000000                          4  * Date       :
00000000                          5  * Description:
00000000                          6  *------------------------------------------------------------
00000000                          7
00000000                          8
00001000                          9  START   ORG     $1000
00001000                         10
00001000                         11
00001000  4E71                   12  JMP_1   NOP
00001002  6000 000E              13          BRA     JMP_4
00001006                         14
00001006  4E71                   15  JMP_2   NOP
00001008  6000 0002              16          BRA     JMP_3
0000100C                         17
0000100C  4E71                   18  JMP_3   NOP
0000100E  6000 0006              19          BRA     DONE
00001012                         20
00001012  4E71                   21  JMP_4   NOP
00001014  60F0                   22          BRA     JMP_2
00001016                         23
00001016  4E71                   24  DONE    NOP
00001018                         25          END     START

No errors detected
```

```
.S68 file read successful
PC=$00001000  CODE=$4E71  NOP
```

PC goes to next instruction

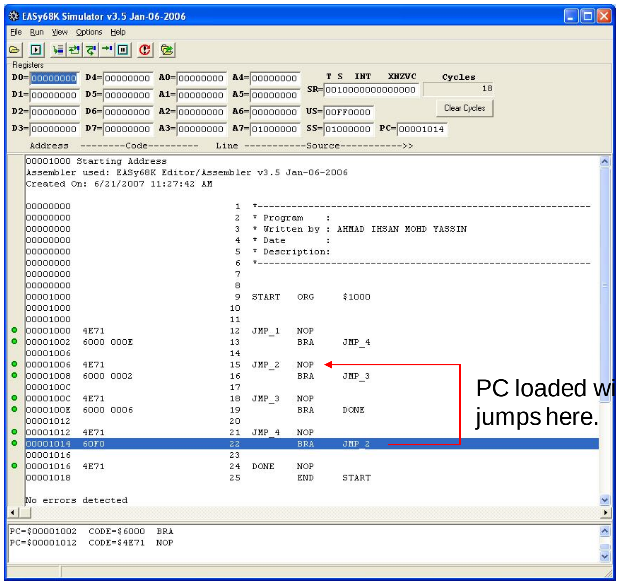
Jumps to $1012 because of branch command.


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          14
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001012
```

```
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 11:27:42 AM

00000000                               1  *-------------------------------------------------------------
00000000                               2  * Program     :
00000000                               3  * Written by  : AHMAD IHSAN MOHD YASSIN
00000000                               4  * Date        :
00000000                               5  * Description:
00000000                               6  *-------------------------------------------------------------
00000000                               7
00000000                               8
00001000                               9  START    ORG     $1000
00001000                              10
00001000                              11
00001000  4E71                        12  JMP_1    NOP
00001002  6000 000E                   13           BRA     JMP_4
00001006                              14
00001006  4E71                        15  JMP_2    NOP
00001008  6000 0002                   16           BRA     JMP_3
0000100C                              17
0000100C  4E71                        18  JMP_3    NOP
0000100E  6000 0006                   19           BRA     DONE
00001012                              20
00001012  4E71                        21  JMP_4    NOP
00001014  60F0                        22           BRA     JMP_2
00001016                              23
00001016  4E71                        24  DONE     NOP
00001018                              25           END     START

No errors detected
```

```
PC=$00001000   CODE=$4E71   NOP
PC=$00001002   CODE=$6000   BRA
```


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          18
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001014
```

```
   Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v3.5 Jan-06-2006
Created On: 6/21/2007 11:27:42 AM

00000000                               1  *-------------------------------------------------------------
00000000                               2  * Program     :
00000000                               3  * Written by  : AHMAD IHSAN MOHD YASSIN
00000000                               4  * Date        :
00000000                               5  * Description:
00000000                               6  *-------------------------------------------------------------
00000000                               7
00000000                               8
00001000                               9  START    ORG     $1000
00001000                              10
00001000                              11
00001000  4E71                        12  JMP_1    NOP
00001002  6000 000E                   13           BRA     JMP_4
00001006                              14
00001006  4E71                        15  JMP_2    NOP
00001008  6000 0002                   16           BRA     JMP_3
0000100C                              17
0000100C  4E71                        18  JMP_3    NOP
0000100E  6000 0006                   19           BRA     DONE
00001012                              20
00001012  4E71                        21  JMP_4    NOP
00001014  60F0                        22           BRA     JMP_2
00001016                              23
00001016  4E71                        24  DONE     NOP
00001018                              25           END     START

No errors detected
```

```
PC=$00001002   CODE=$6000   BRA
PC=$00001012   CODE=$4E71   NOP
```

PC loaded with $1006, jumps here.

Stack Pointer

Software Stack

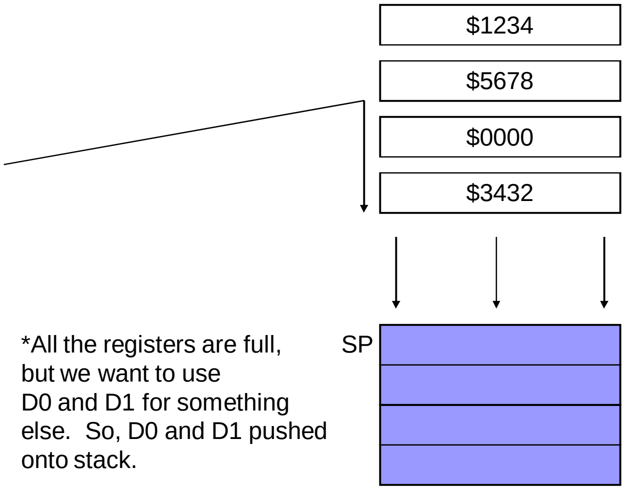
- Sometimes, registers not enough to store data:
  - □ Only 8 data registers available.
- Solution: reserve part of memory for storing data – software stack.
- Data can be stored/retrieved by pushing/popping.

Software Stack

- Stacks are usually used to store:
  - □ Data register values.
  - □ Program Counter.
  - □ Status Register.
  - □ Other information.
- Stacks operate on LIFO basis (Last In First Out).
- Pushes (puts) items on stack and pops (takes) in reverse order.

The Software Stack - Push

D0	$12345678
D1	$00003432
D2	…
D3	…
D4	…
D5	…
D6	…
D7	…



$1234

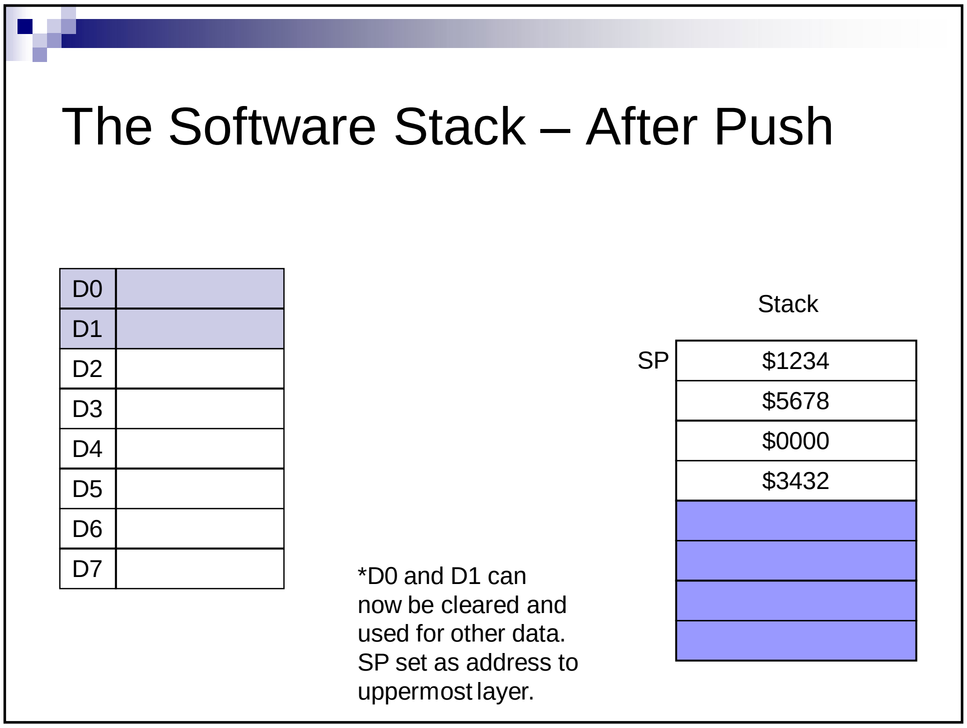
$5678

$0000

$3432

SP

*All the registers are full, but we want to use D0 and D1 for something else. So, D0 and D1 pushed onto stack.



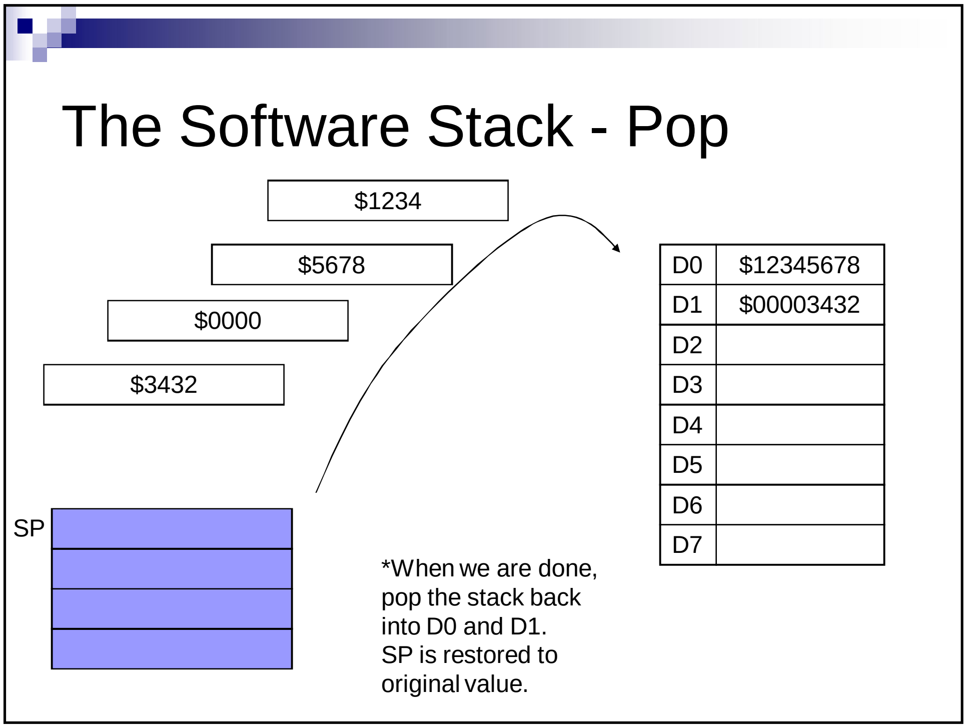
The Software Stack – After Push

D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

Stack

SP	$1234
	$5678
	$0000
	$3432

*D0 and D1 can now be cleared and used for other data. SP set as address to uppermost layer.



The Software Stack - Pop

D0	$12345678
D1	$00003432
D2	
D3	
D4	
D5	
D6	
D7	

*When we are done, pop the stack back into D0 and D1. SP is restored to original value.

Stack Pointer (SP)

- Stack pointer: special register to keep record of stack use:
  - Keeps track of top-most stack position.
- A7 used to store SP.
- 32-bits: large enough to address all memory locations.

Stack Pointer in Action – Moving Data into Stack

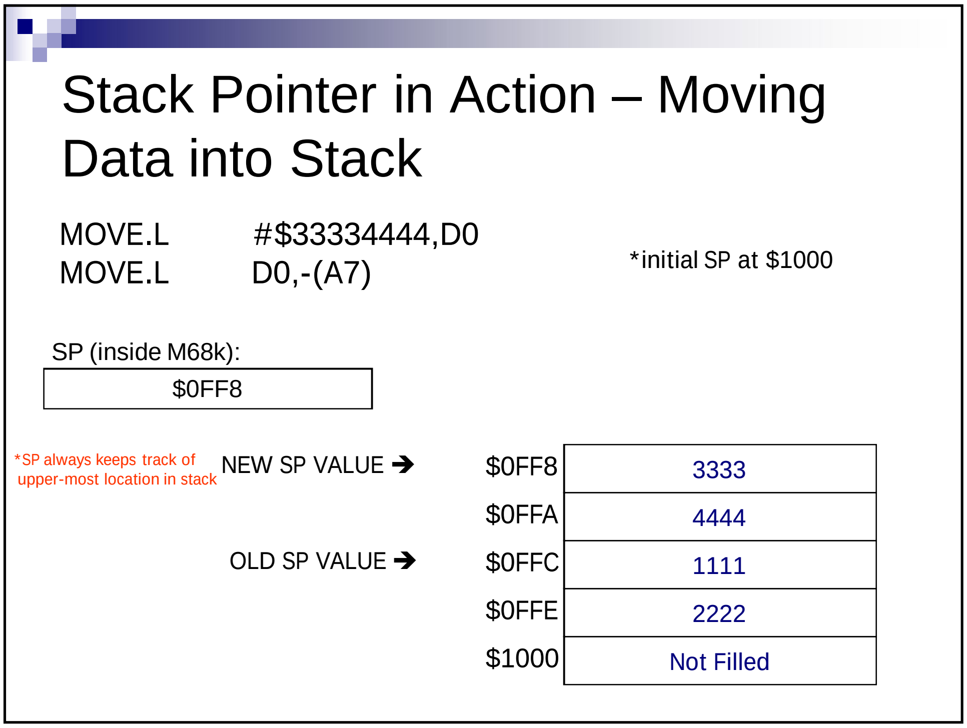
```
MOVE.L      #$11112222,D0
MOVE.L      D0,-(A7)
```

*initial SP at $1000

SP (inside M68k):

$0FFC

	$0FF8	
	$0FFA	
NEW SP VALUE ➔	$0FFC	1111
	$0FFE	2222
OLD SP VALUE ➔	$1000	Not Filled



Stack Pointer in Action – Moving Data into Stack

```
MOVE.L      #$33334444,D0
MOVE.L      D0,-(A7)
```

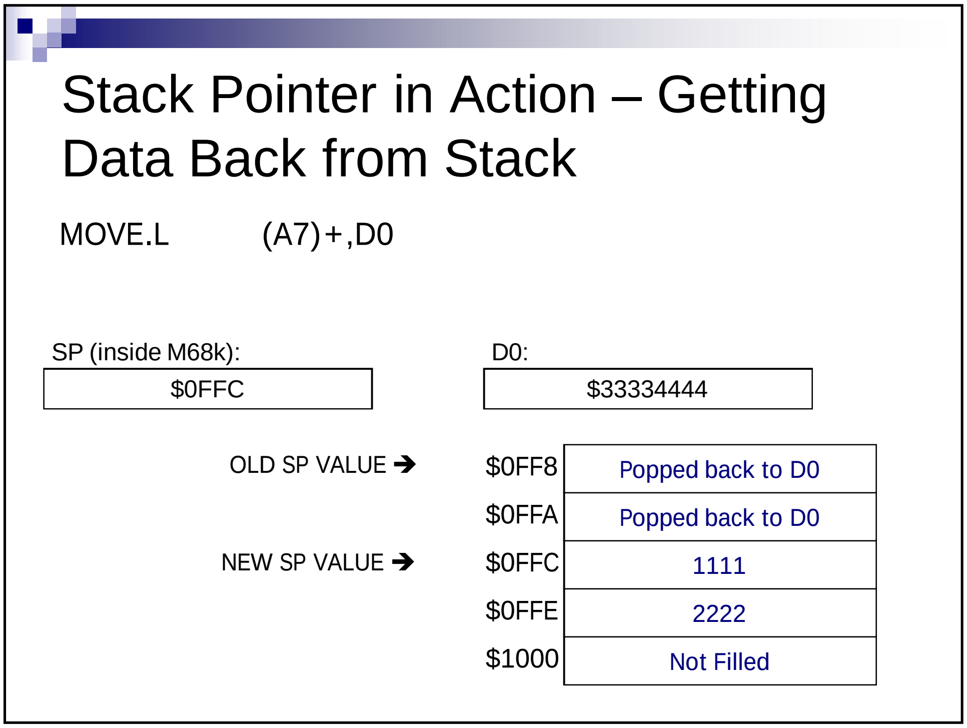
*initial SP at $1000

SP (inside M68k):

$0FF8

*SP always keeps track of upper-most location in stack

NEW SP VALUE ➔	$0FF8	3333
	$0FFA	4444
OLD SP VALUE ➔	$0FFC	1111
	$0FFE	2222
	$1000	Not Filled



Stack Pointer in Action – Getting Data Back from Stack

```
MOVE.L       (A7)+,D0
```

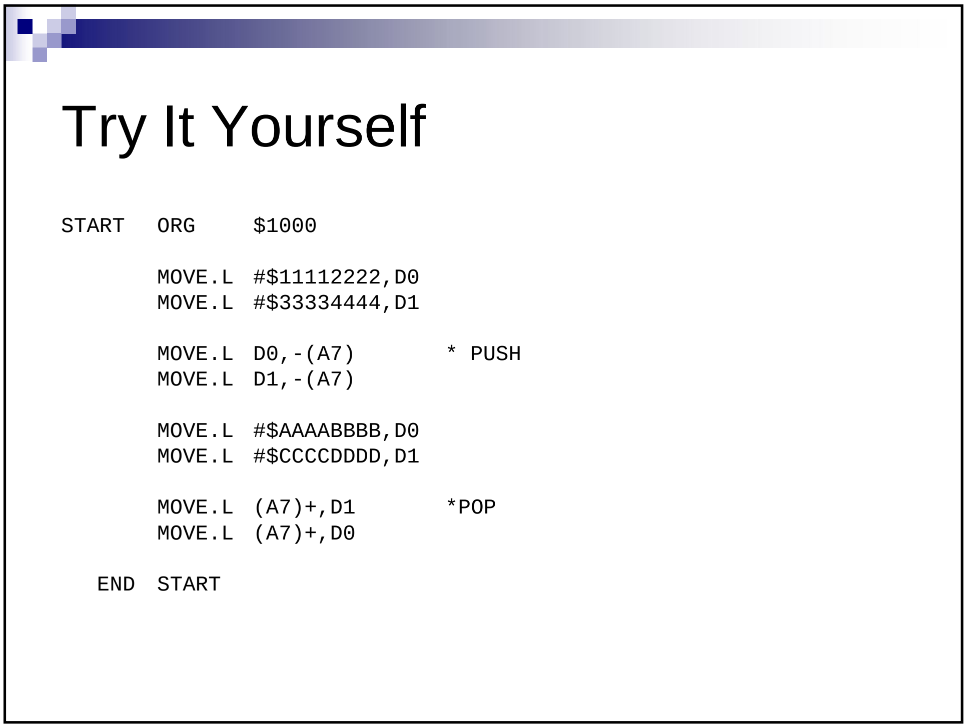
SP (inside M68k):

$0FFC

D0:

$33334444

OLD SP VALUE ➔	$0FF8	Popped back to D0
	$0FFA	Popped back to D0
NEW SP VALUE ➔	$0FFC	1111
	$0FFE	2222
	$1000	Not Filled



Try It Yourself

```
START   ORG     $1000

        MOVE.L  #$11112222,D0
        MOVE.L  #$33334444,D1

        MOVE.L  D0,-(A7)        * PUSH
        MOVE.L  D1,-(A7)

        MOVE.L  #$AAAABBBB,D0
        MOVE.L  #$CCCCDDDD,D1

        MOVE.L  (A7)+,D1        *POP
        MOVE.L  (A7)+,D0

   END  START
```

Status Register

Status Register

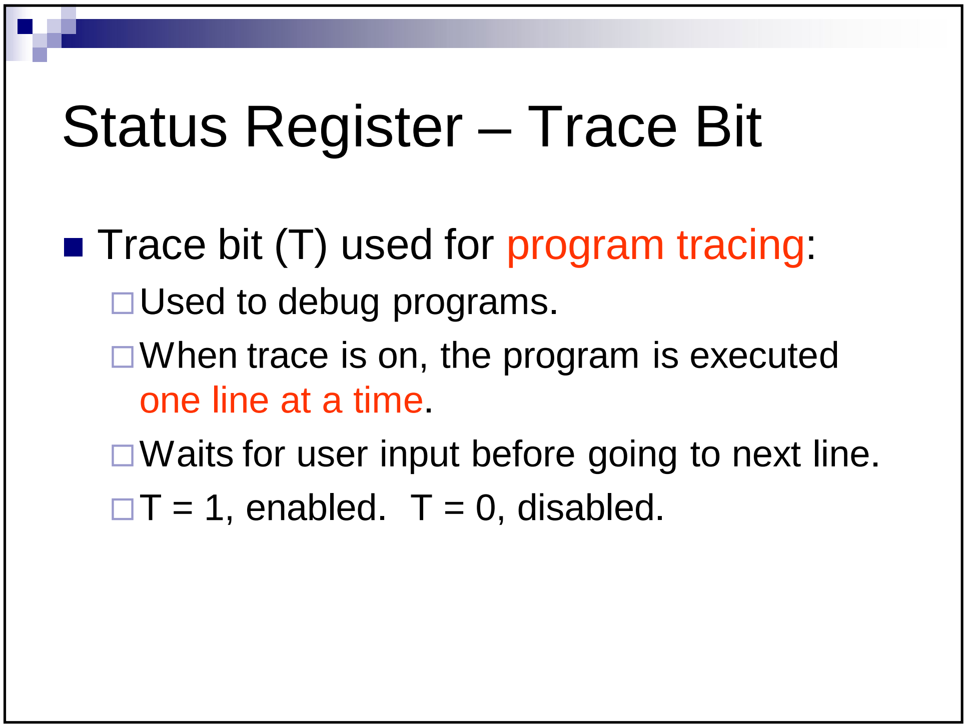
- Special purpose register, 16-bits wide.
- Stores important control and status bits of arithmetic calculations.
- Consists of:
  - □ Trace flag.
  - □ Supervisor flags.
  - □ Interrupt mask flags.
  - □ Condition Code Register (CCR) flags.

The Status Register

T		S			I_2	I_1	I_0				X	N	Z	V	C

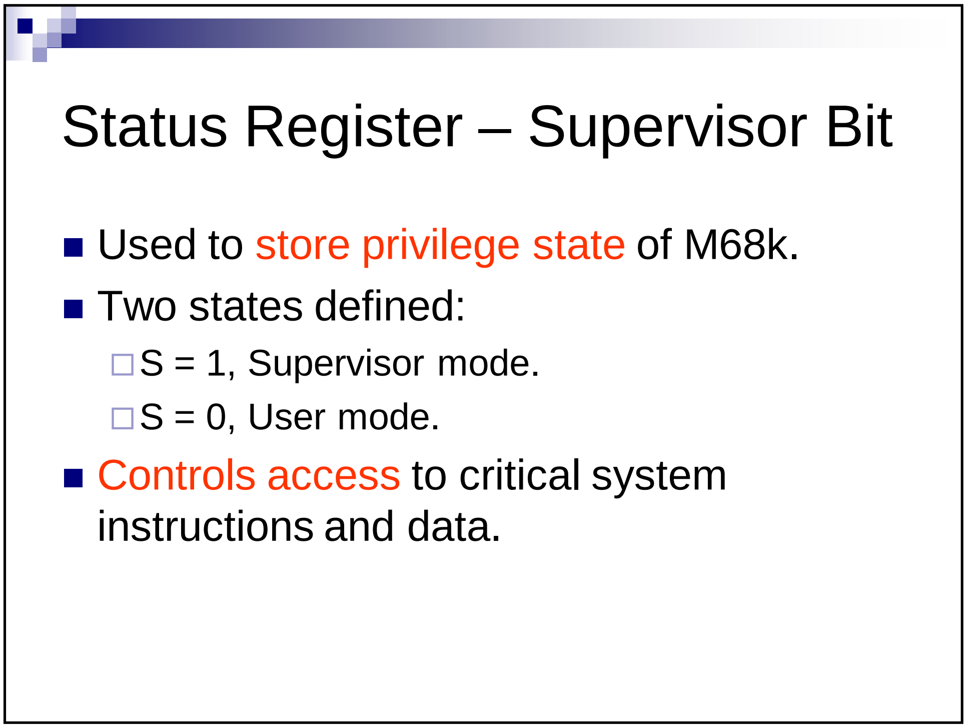
CCR

T = Trace Flag.
S = Supervisor Flag.
I_2, I_1, I_0 = Interrupt Mask Flags.
X = Extend Flag.
N = Negative Flag.
Z = Zero Flag.
V = Overflow Flag.
C = Carry Flag.



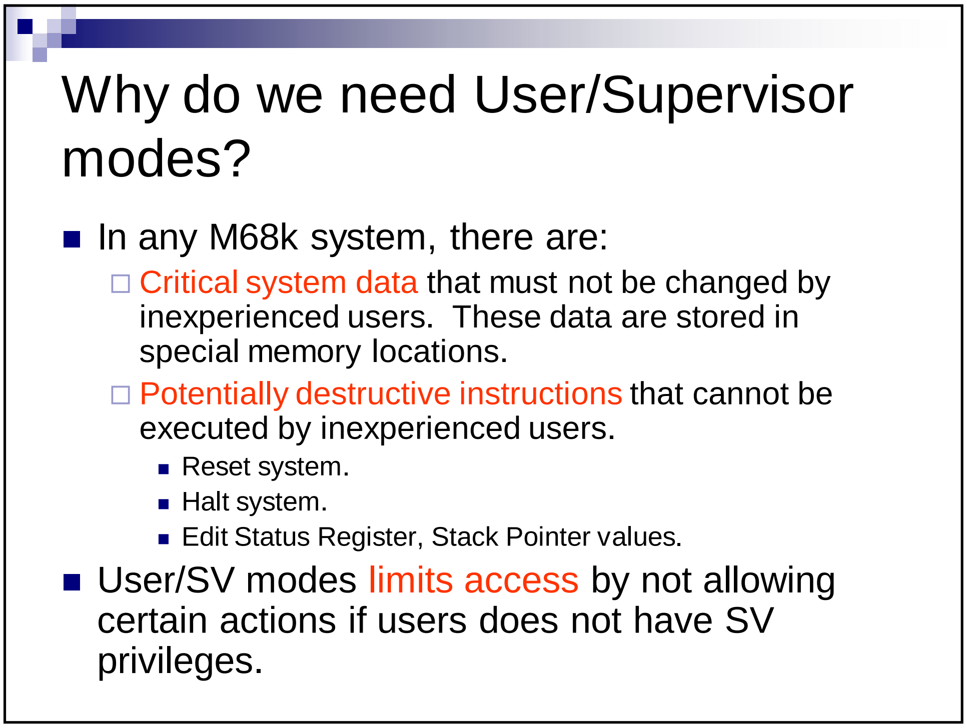
Status Register – Trace Bit

- Trace bit (T) used for program tracing:
  - □Used to debug programs.
  - □When trace is on, the program is executed one line at a time.
  - □Waits for user input before going to next line.
  - □T = 1, enabled. T = 0, disabled.



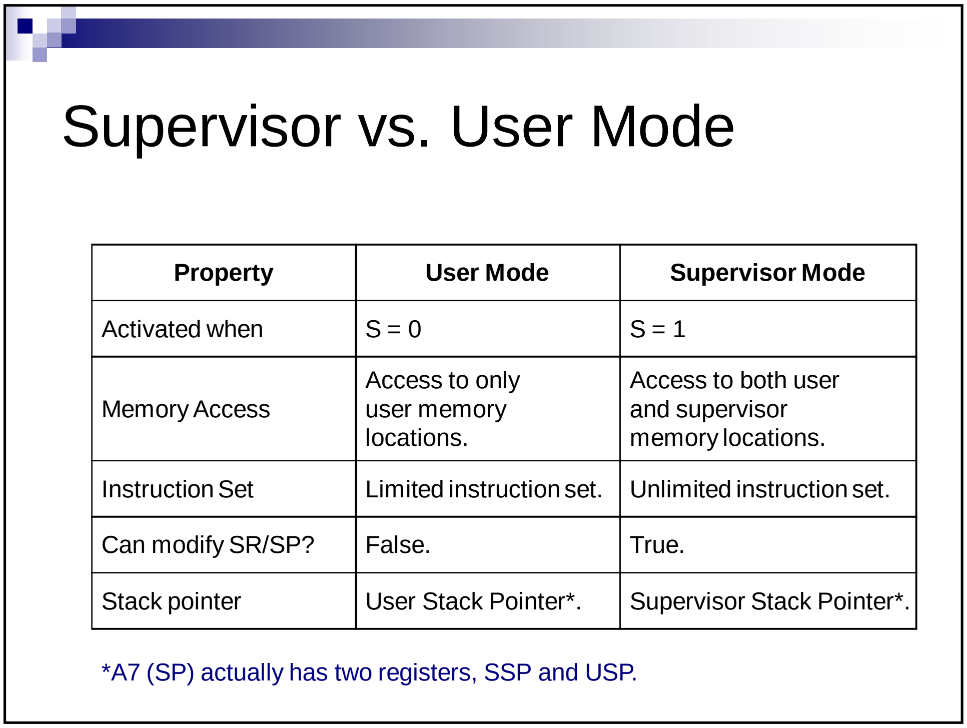
Status Register – Supervisor Bit

- Used to store privilege state of M68k.
- Two states defined:
  - □S = 1, Supervisor mode.
  - □S = 0, User mode.
- Controls access to critical system instructions and data.



Why do we need User/Supervisor modes?

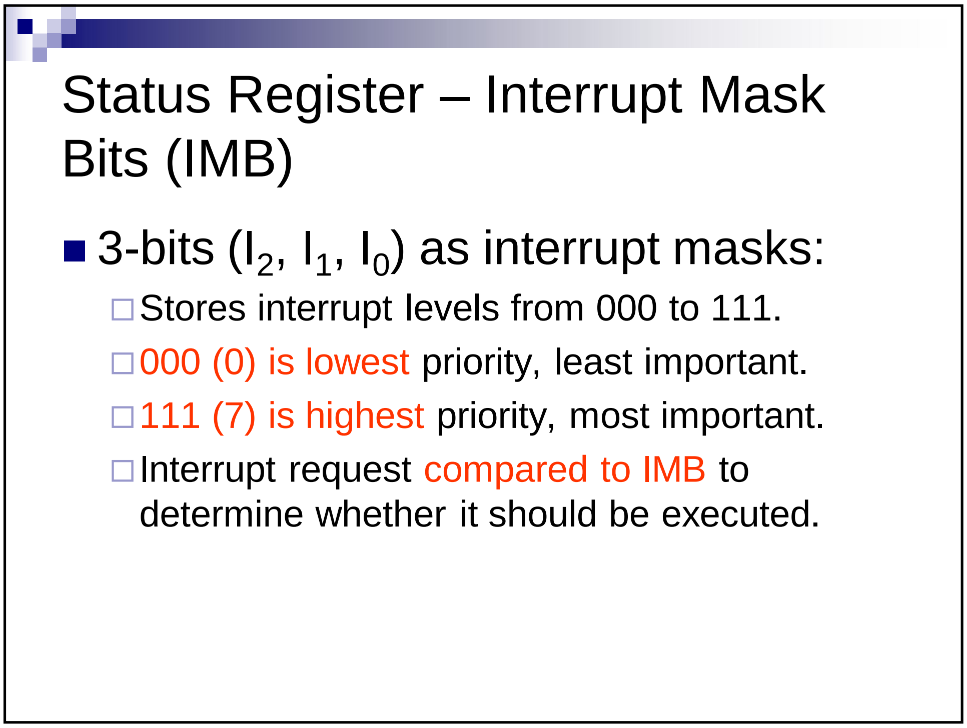
- In any M68k system, there are:
  - Critical system data that must not be changed by inexperienced users. These data are stored in special memory locations.
  - Potentially destructive instructions that cannot be executed by inexperienced users.
    - Reset system.
    - Halt system.
    - Edit Status Register, Stack Pointer values.
- User/SV modes limits access by not allowing certain actions if users does not have SV privileges.



Supervisor vs. User Mode

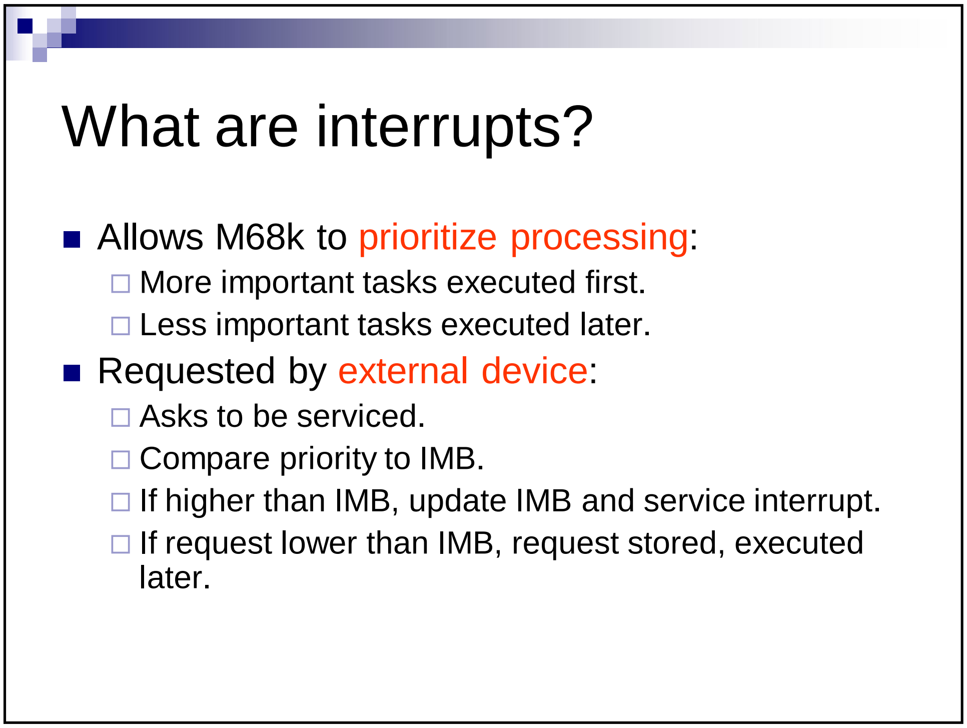
Property	User Mode	Supervisor Mode
Activated when	S = 0	S = 1
Memory Access	Access to only user memory locations.	Access to both user and supervisor memory locations.
Instruction Set	Limited instruction set.	Unlimited instruction set.
Can modify SR/SP?	False.	True.
Stack pointer	User Stack Pointer*.	Supervisor Stack Pointer*.

*A7 (SP) actually has two registers, SSP and USP.



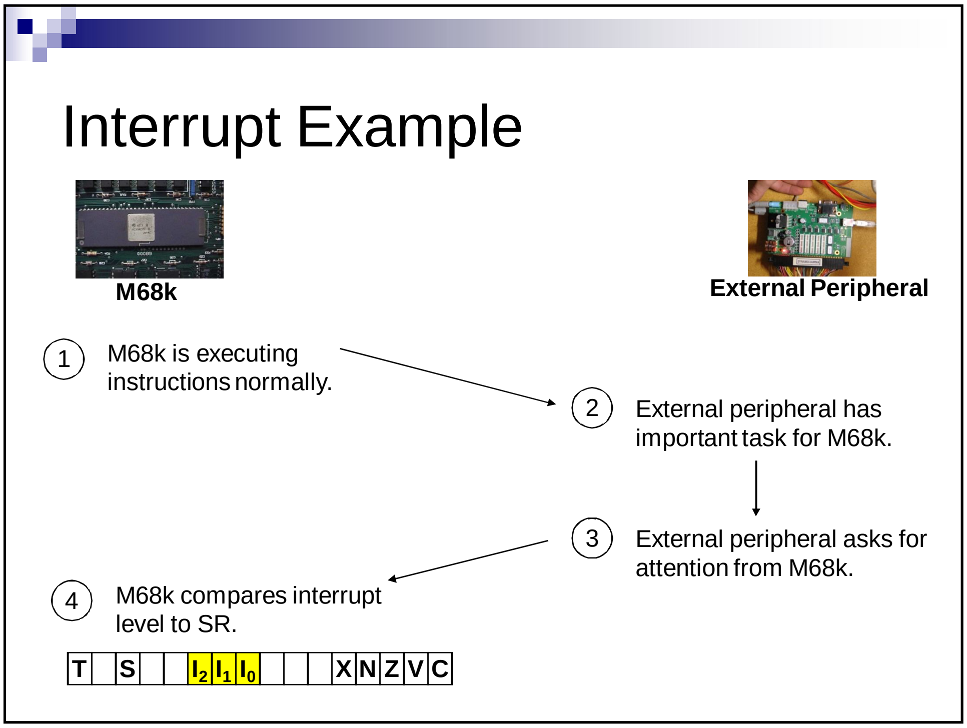
Status Register – Interrupt Mask Bits (IMB)

- 3-bits (I_2, I_1, I_0) as interrupt masks:
  - □ Stores interrupt levels from 000 to 111.
  - □ 000 (0) is lowest priority, least important.
  - □ 111 (7) is highest priority, most important.
  - □ Interrupt request compared to IMB to determine whether it should be executed.



What are interrupts?

- Allows M68k to prioritize processing:
  - □ More important tasks executed first.
  - □ Less important tasks executed later.
- Requested by external device:
  - □ Asks to be serviced.
  - □ Compare priority to IMB.
  - □ If higher than IMB, update IMB and service interrupt.
  - □ If request lower than IMB, request stored, executed later.



Interrupt Example

M68k

External Peripheral

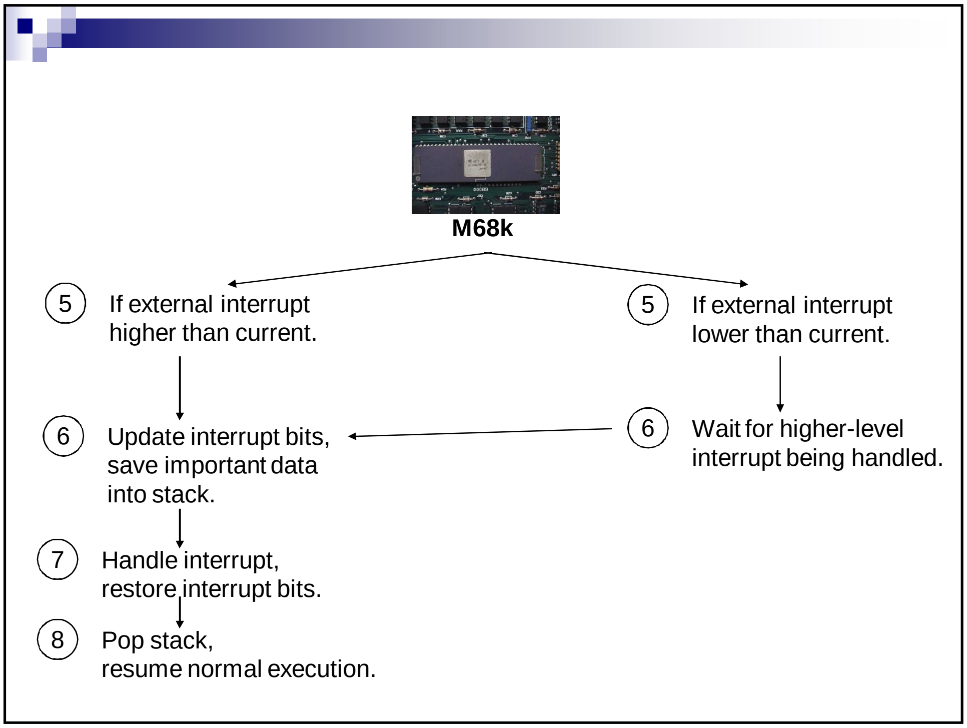
1 M68k is executing instructions normally.

2 External peripheral has important task for M68k.

3 External peripheral asks for attention from M68k.

4 M68k compares interrupt level to SR.

T		S			I_2	I_1	I_0				X	N	Z	V	C



M68k

5 If external interrupt higher than current.

5 If external interrupt lower than current.

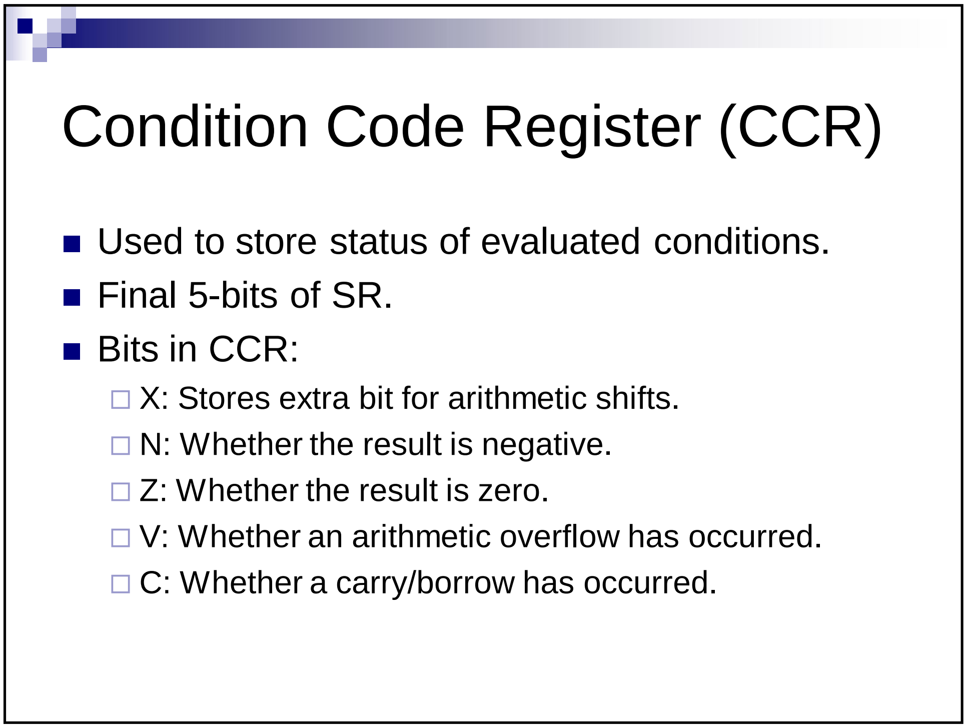
6 Wait for higher-level interrupt being handled.

6 Update interrupt bits, save important data into stack.

7 Handle interrupt, restore interrupt bits.

8 Pop stack, resume normal execution.

Condition Code Register (CCR)



Condition Code Register (CCR)

- Used to store status of evaluated conditions.
- Final 5-bits of SR.
- Bits in CCR:
  - X: Stores extra bit for arithmetic shifts.
  - N: Whether the result is negative.
  - Z: Whether the result is zero.
  - V: Whether an arithmetic overflow has occurred.
  - C: Whether a carry/borrow has occurred.

X: Extend Bit

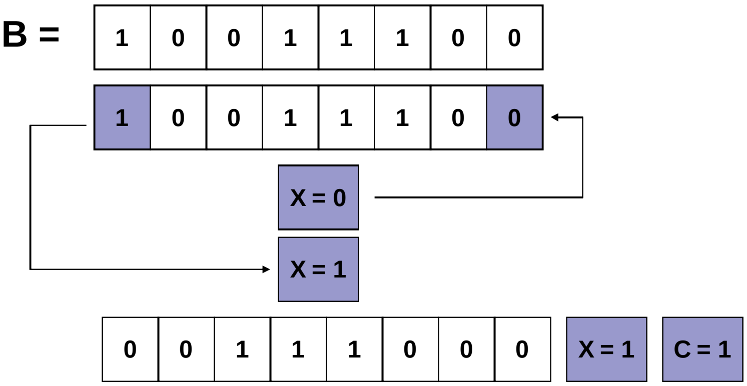
- Purpose:
  - □ To allow rotate and shift operations:
    - Stores the extra shifted bit during rotation.
  - □ For multi-precision arithmetic in BCD operations.
  - □ Usually set according to C.

X Example – Rotate

```
MOVE.B   #$9C,D0
ROXL.B   #1,D0
```

* Assuming X = 0 before ROXL

D0.B =



1	0	0	1	1	1	0	0
1	0	0	1	1	1	0	0

X = 0

X = 1

0	0	1	1	1	0	0	0	X = 1	C = 1

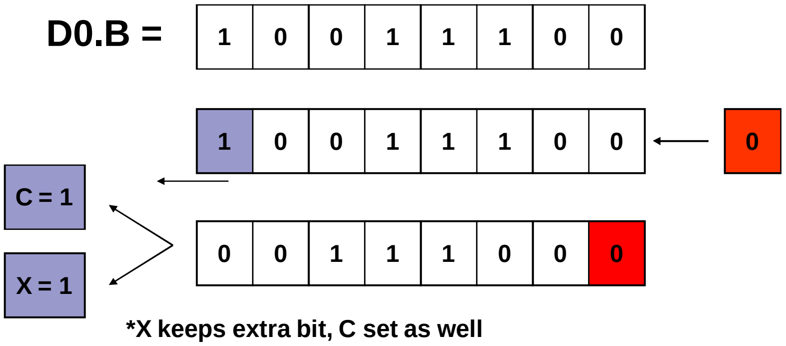
C set as well

Try It Yourself

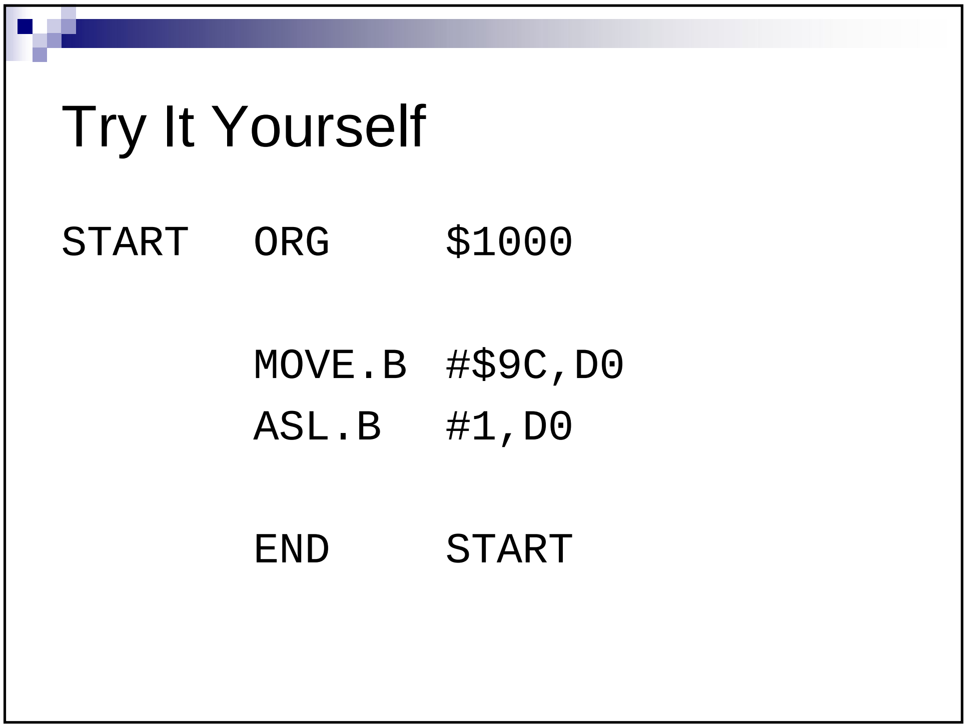
```
START   ORG     $1000

        MOVE.B  #$9C,D0
        ROXL.B  #1,D0

END     START
```

X Example – Arithmetic Shift

ASL.B #1,D0

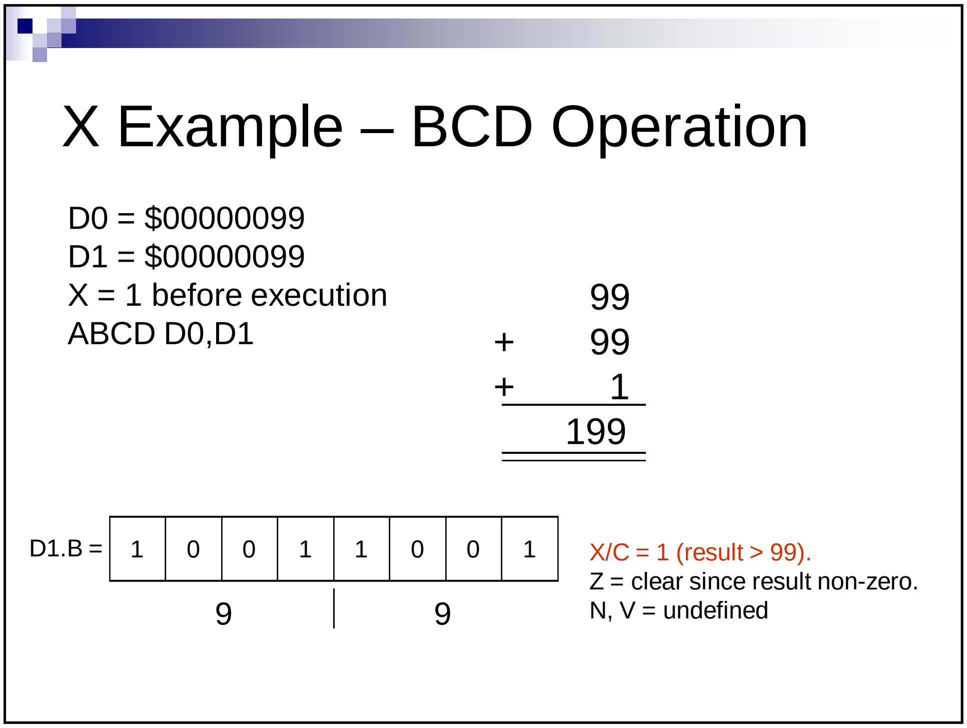


***X keeps extra bit, C set as well**



Try It Yourself

```
START   ORG     $1000

        MOVE.B  #$9C,D0
        ASL.B   #1,D0

        END     START
```



X Example – BCD Operation

D0 = $00000099
D1 = $00000099
X = 1 before execution
ABCD D0,D1

```
      99
+     99
+      1
     199
```

D1.B =	1	0	0	1	1	0	0	1
	9				9			

X/C = 1 (result > 99).
Z = clear since result non-zero.
N, V = undefined

Try It Yourself

```
START    ORG      $1000

         MOVE.B   #$99,D0
         MOVE.B   #$99,D1

         ABCD     D0,D1

         END      START
```

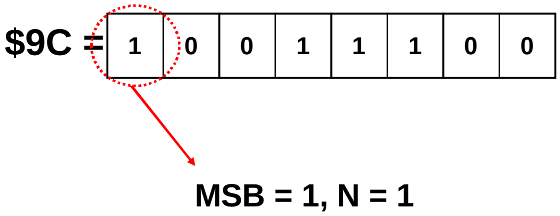
N – Negative Bit

- Purpose: to test whether the result is negative.
- Does this by examining the MSB:
  - □ In 2's complement, MSB is sign bit.
  - □ If MSB = 0, N = 0 (not negative).
  - □ If MSB = 1, N = 1 (negative).

N Example

- MOVE.B #0,D0
- MOVE.B #100,D1
- SUB.B D1,D0

D0 = $00 (0)
D1 = $64 (100)
D0 = 0 – 100 = $9C (-100)



$9C =

1	0	0	1	1	1	0	0

MSB = 1, N = 1

Try It Yourself

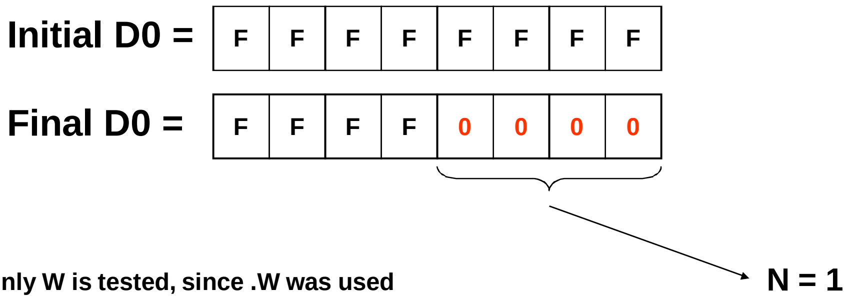
```
START   ORG     $1000

        MOVE.B  #0,D0
        MOVE.B  #100,D1
        SUB.B   D1,D0

        END     START
```

Z – Zero Bit

- Purpose: to test whether the result is zero.
- Does this by examining all active bits.
  - □ If all active bits = 0, Z = 1 (is zero).
  - □ If not all active bits = 0, Z = 0 (not zero).

Z Example

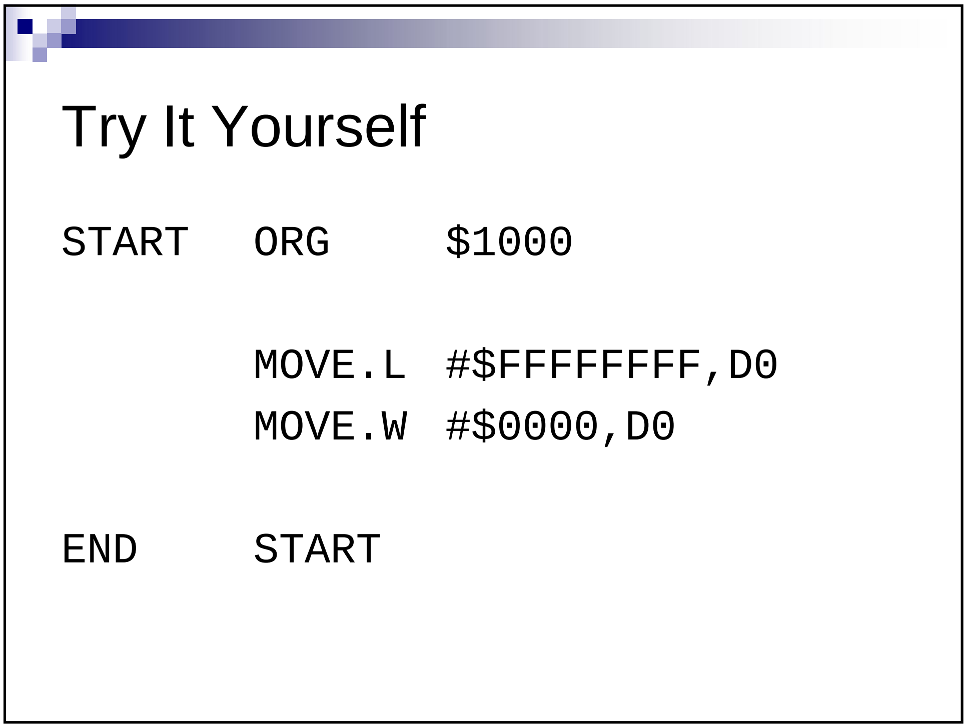
- MOVE.L #$FFFFFFFF,D0
- MOVE.W #$0000,D0



Initial D0 =	F	F	F	F	F	F	F	F
Final D0 =	F	F	F	F	0	0	0	0

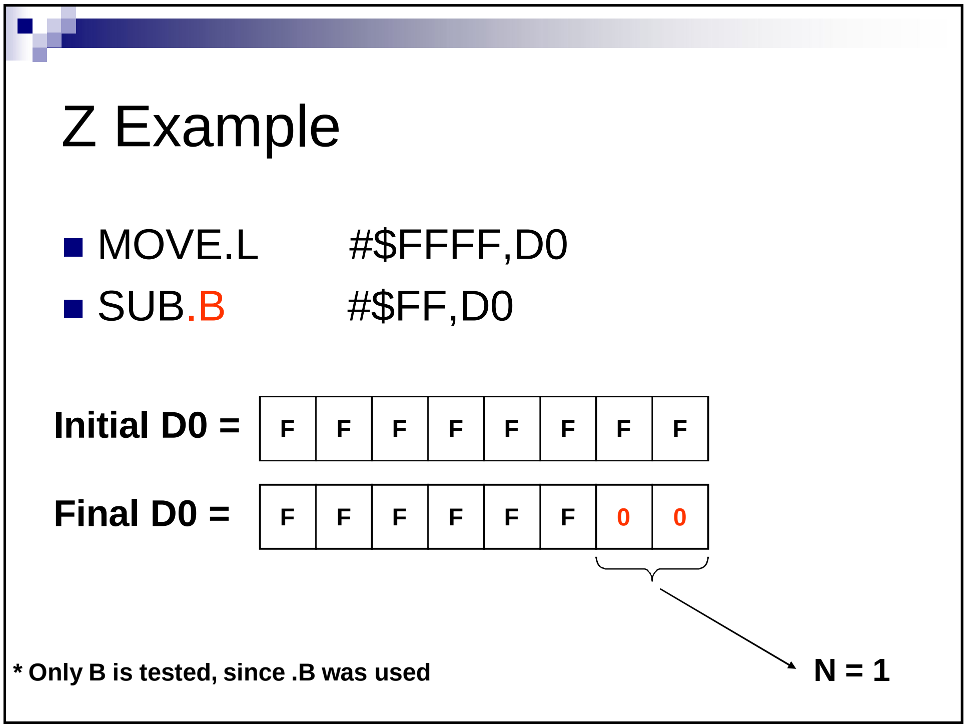
N = 1

*** Only W is tested, since .W was used**



Try It Yourself

```
START   ORG     $1000

        MOVE.L  #$FFFFFFFF,D0
        MOVE.W  #$0000,D0

END     START
```



Z Example

- MOVE.L #$FFFF,D0
- SUB.B #$FF,D0

Initial D0 =	F	F	F	F	F	F	F	F
Final D0 =	F	F	F	F	F	F	0	0

N = 1

*** Only B is tested, since .B was used**

Try It Yourself

```
START   ORG     $1000

        MOVE.L  #$FFFFFFFF,D0
        SUB.B   #$FF,D0

        END     START
```

V – Overflow Bit

- Checks whether sign bit is changed as a result of arithmetic operation.
  - Set when arithmetic operation changes MSB.
  - Only few instructions can change V.
  - Check with M68k instruction reference.

V- Overflow Bit

- In ADD, SUB and CMP, V is modified differently.
- For ADD, SUB & CMP:
  - □ P + P = N
  - □ N + N = P
  - □ P – N = N
  - □ N – P = P

* P = Positive Number (MSB = 0)
N = Negative Number (MSB = 1)

V Example

- MOVE.B #$41,D0
- MOVE.B #$46,D1
- ADD.B D1,D0

D0 = 65 + 70

										2's complement
	D0.B =	0	1	0	0	0	0	0	1	+65
+	D1.B =	0	1	0	0	0	1	1	0	+70
	New D0.B =	1	0	0	0	0	1	1	1	-121?

V = 1

Try It Yourself

```
START    ORG       $1000

         MOVE.B    #$41,D0
         MOVE.B    #$46,D1

         ADD.B     D1,D0

         END       START
```

V Example

- MOVE.B #$DE,D0
- MOVE.B #$9F,D1
- ADD.B   D1,D0

D0 = -34 + (-97)

										2's complement
	D0.B =	1	1	0	1	1	1	1	0	-34
+	D1.B =	1	0	0	1	1	1	1	1	-97
	New D0.B =	0	1	1	1	1	1	0	1	+125?

V = 1

Try It Yourself

```
START     ORG       $1000

          MOVE.B    #$DE,D0
          MOVE.B    #$9F,D1

          ADD.B     D1,D0

          END       START
```

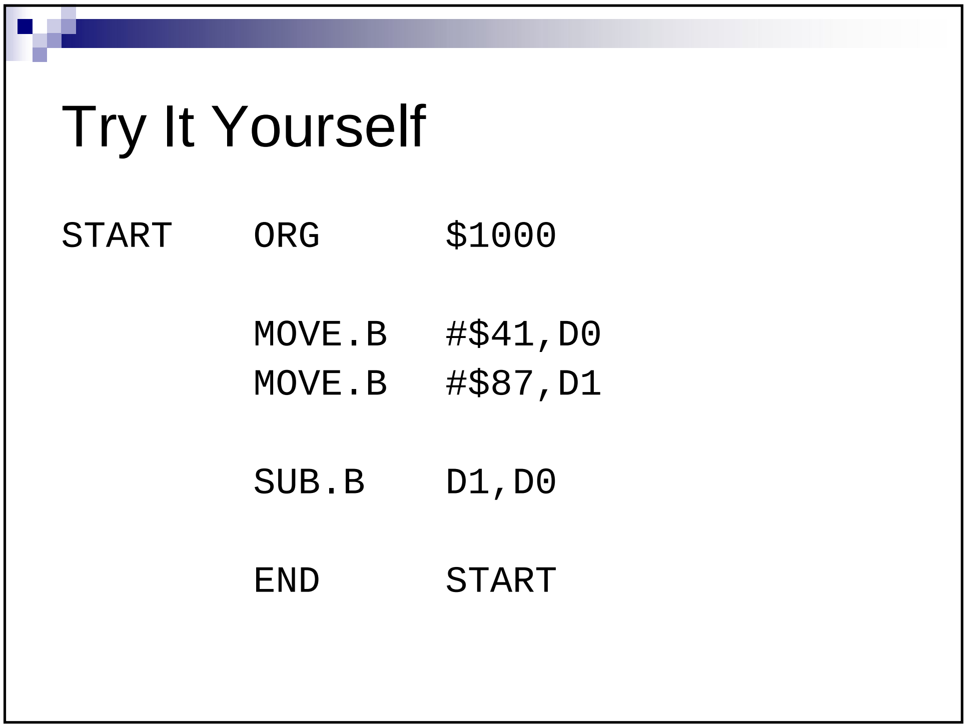
V Example

- MOVE.B #$41,D0
- MOVE.B #$87,D1
- SUB.B D1,D0

D0 = +65 – (-121)

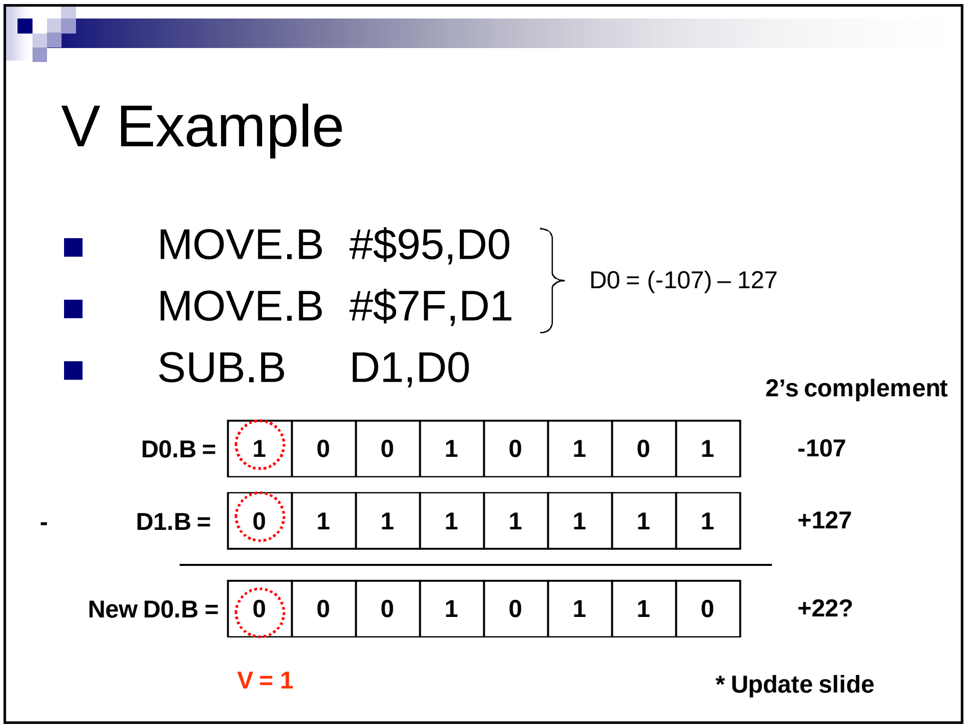
									2's complement
D0.B =	0	1	0	0	0	0	0	1	+65
- D1.B =	1	0	0	0	0	1	1	1	-121
New D0.B =	1	0	1	1	1	0	1	0	-70?

V = 1



Try It Yourself

```
START    ORG      $1000

         MOVE.B   #$41,D0
         MOVE.B   #$87,D1

         SUB.B    D1,D0

         END      START
```



V Example

- MOVE.B #$95,D0
- MOVE.B #$7F,D1
- SUB.B D1,D0

D0 = (-107) – 127

									2's complement
D0.B =	1	0	0	1	0	1	0	1	-107
- D1.B =	0	1	1	1	1	1	1	1	+127
New D0.B =	0	0	0	1	0	1	1	0	+22?

V = 1

* Update slide

Try It Yourself

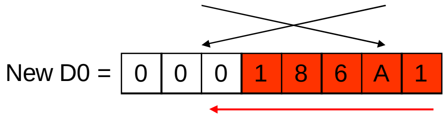
```
START     ORG       $1000

          MOVE.B    #$95,D0
          MOVE.B    #$7F,D1

          SUB.B     D1,D0

          END       START
```

DIVU Example - Overflow

D0 = $00186A1 (100,001 decimal)
D1 = $0000001 (1 decimal)
DIVU   D1,D0

100,001 / 1 = 100,001 = $186A1

	Quotient	Remainder
Decimal	100,001	0
Hex	$186A1	$0000



New D0 =	0	0	0	1	8	6	A	1

V = 1 (divide overflow)

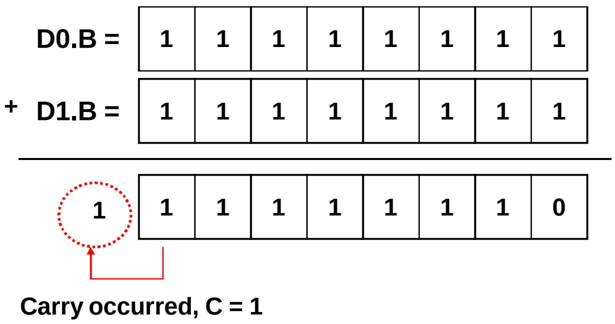
Quotient overflowed to remainder.

C – Carry Bit

- Purpose: to test whether a borrow/carry has occurred:
  - □ Carry occurs when addition results are “carried forward”.
  - □ Borrow occurs when subtraction results are “borrowed”.
- Does this by examining the results of addition/subtraction operations.

C Example - Carry

- MOVE.B #$FF,D0
- MOVE.B #$FF,D1
- ADD.B D1,D0



D0.B =	1	1	1	1	1	1	1	1
+ D1.B =	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0

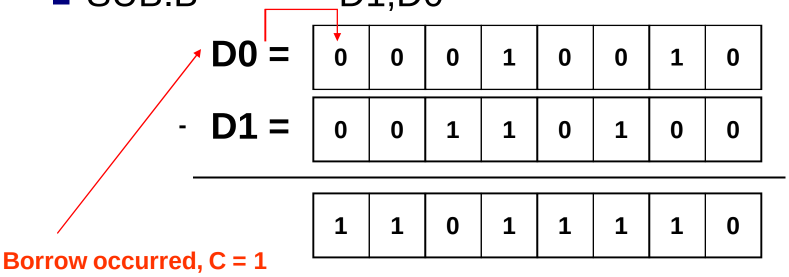
Carry occurred, C = 1

Try It Yourself

```
START     ORG       $1000

          MOVE.B    #$FF,D0
          MOVE.B    #$FF,D1

          ADD.B     D1,D0

          END       START
```

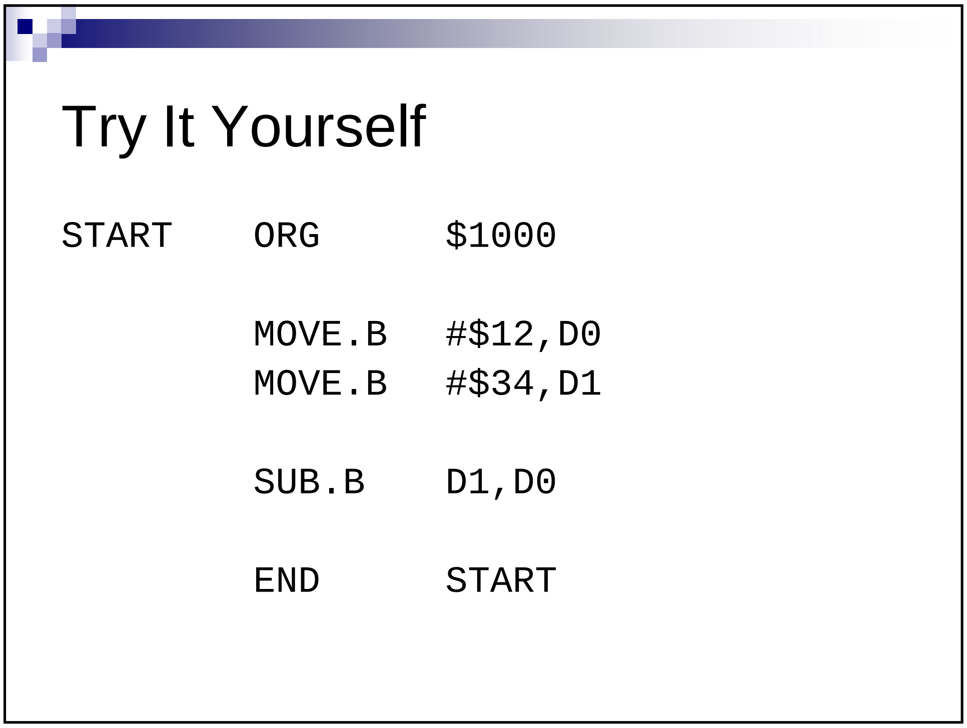
C Example - Borrow

- MOVE.B #$12,D0
- MOVE.B #$34,D1
- SUB.B D1,D0



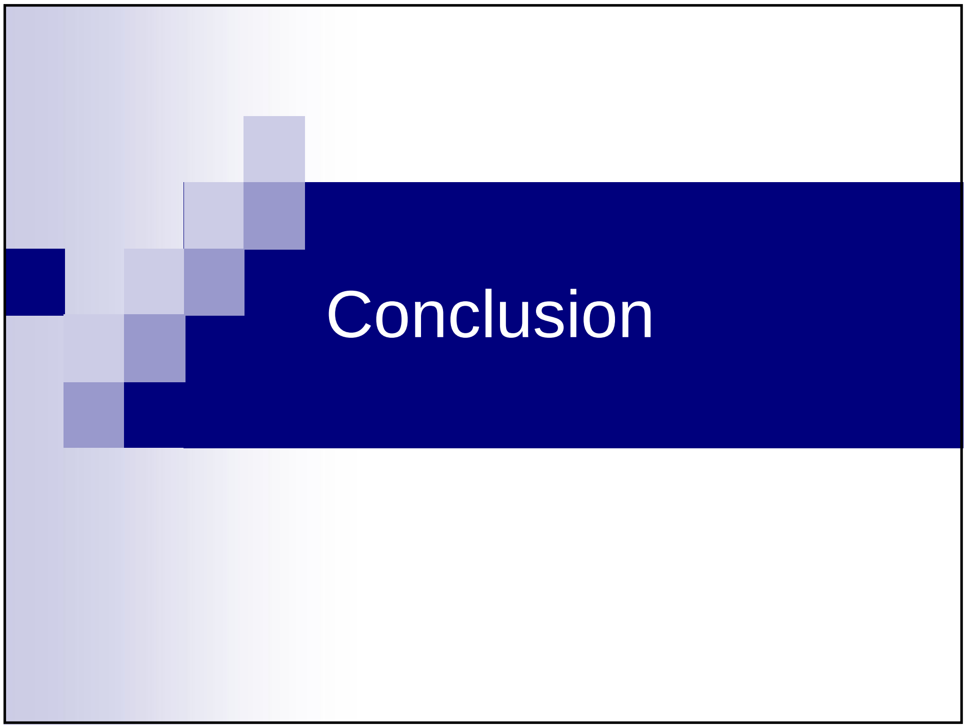
D0 =	0	0	0	1	0	0	1	0
- D1 =	0	0	1	1	0	1	0	0
	1	1	0	1	1	1	1	0

Borrow occurred, C = 1

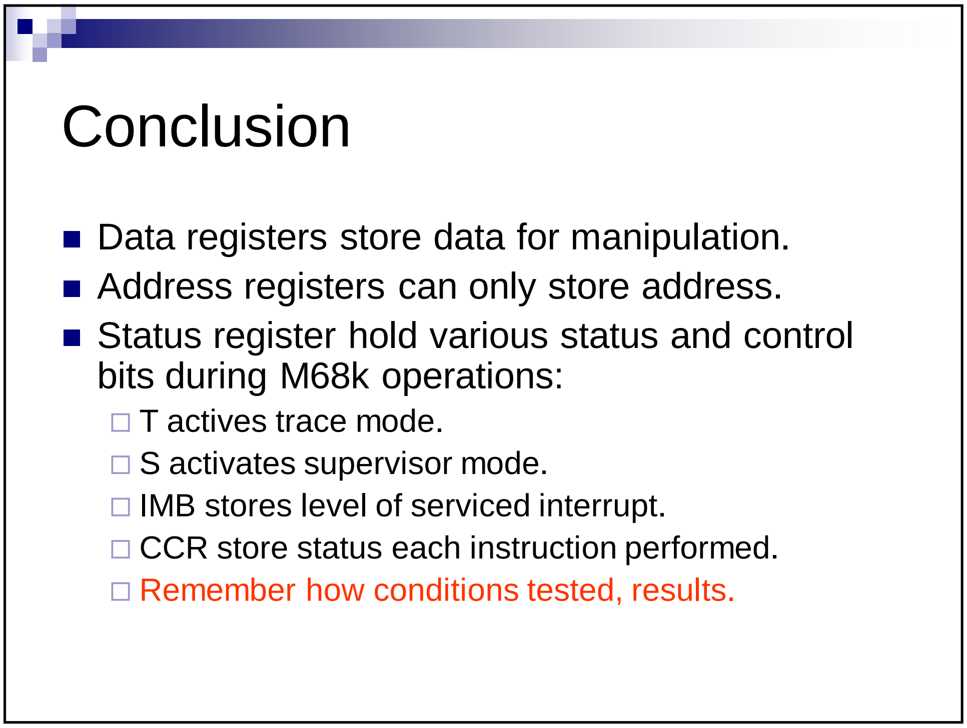


Try It Yourself

```
START    ORG      $1000

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

         SUB.B    D1,D0

         END      START
```

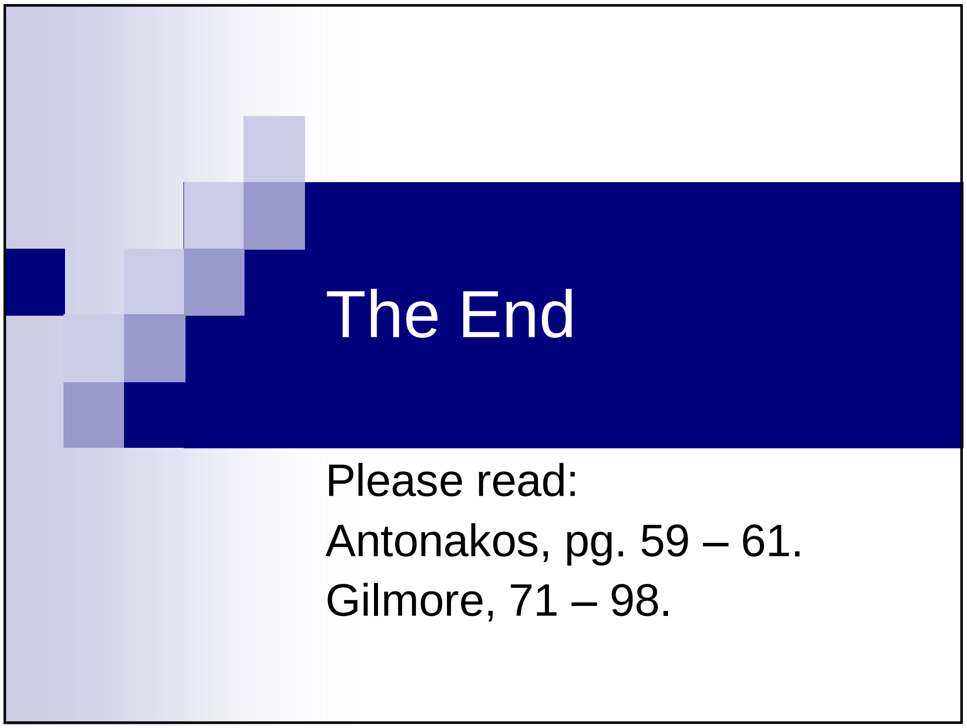


Conclusion



Conclusion

- Data registers store data for manipulation.
- Address registers can only store address.
- Status register hold various status and control bits during M68k operations:
  - T actives trace mode.
  - S activates supervisor mode.
  - IMB stores level of serviced interrupt.
  - CCR store status each instruction performed.
  - Remember how conditions tested, results.



The End

Please read:
Antonakos, pg. 59 – 61.
Gilmore, 71 – 98.