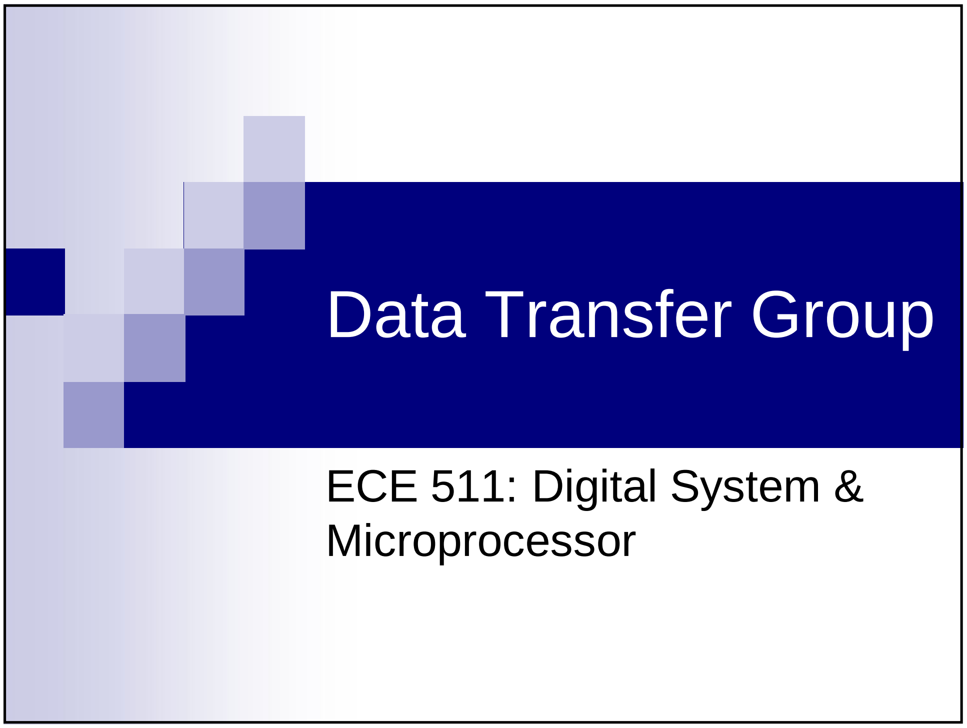
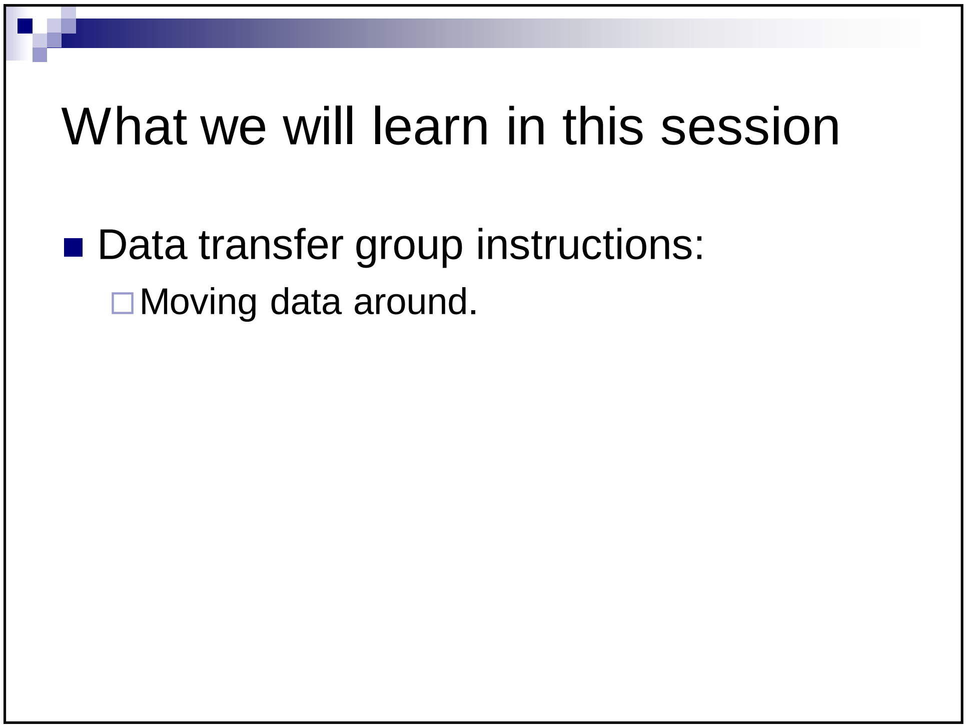


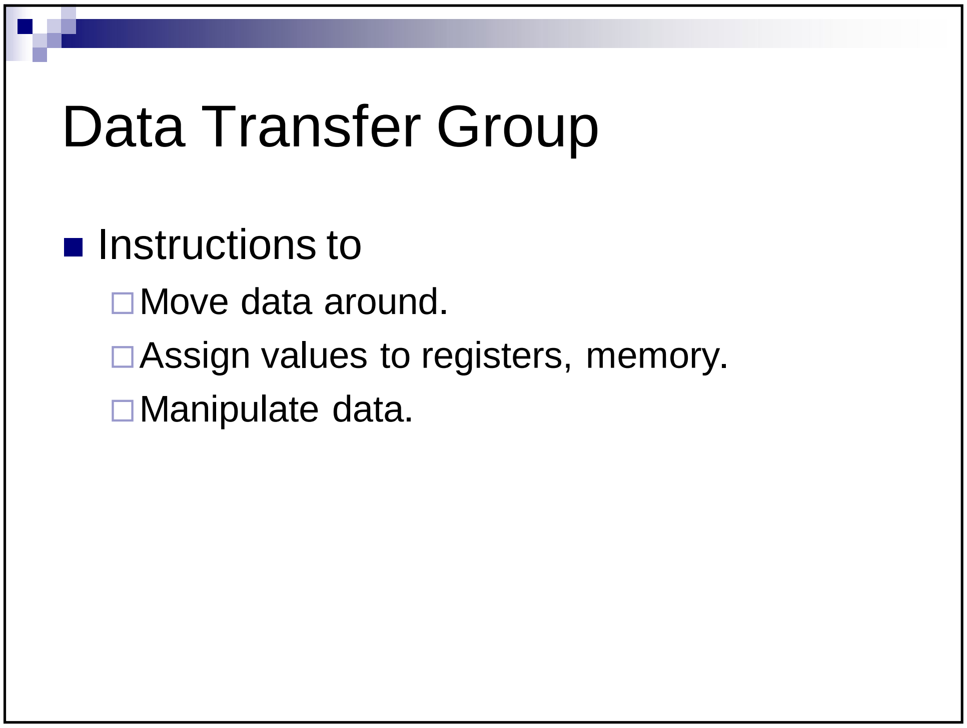
Data Transfer Group

ECE 511: Digital System & Microprocessor



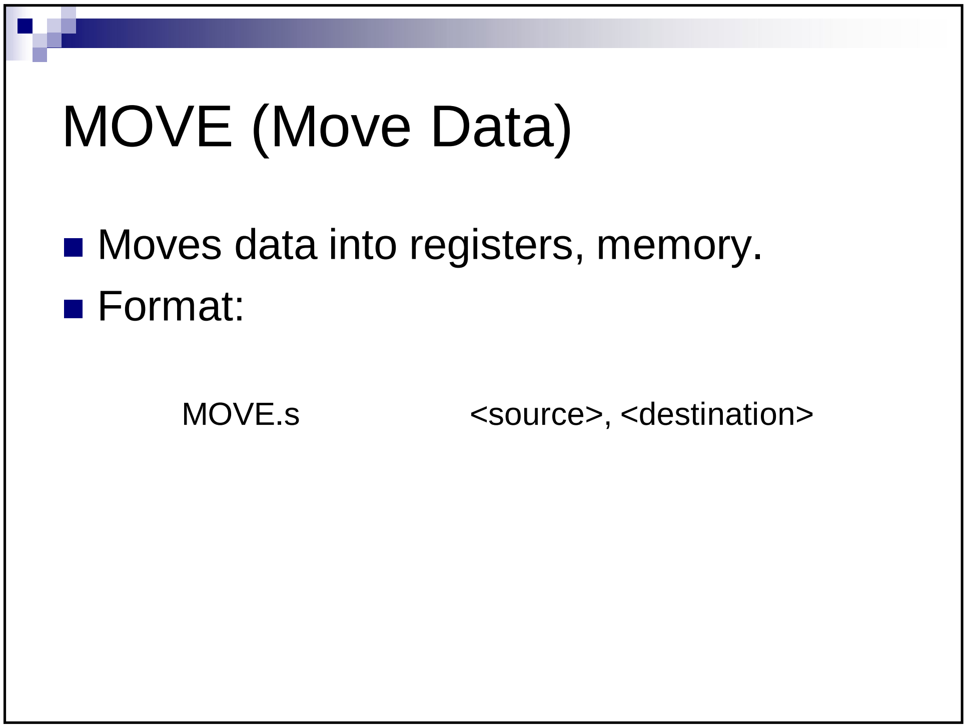
What we will learn in this session

- Data transfer group instructions:
  - Moving data around.



Data Transfer Group

- Instructions to
  - □ Move data around.
  - □ Assign values to registers, memory.
  - □ Manipulate data.



MOVE (Move Data)

- Moves data into registers, memory.
- Format:

```
MOVE.s          <source>, <destination>
```

MOVE

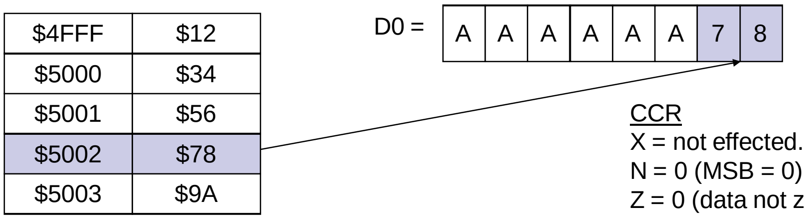
Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	s	s	s	s	s	s	s	s	s	s	s
d	-	d	d	d	d	d	d	d	-	-	-

X	N	Z	V	C
-	*	*	0	0

BWL

MOVE Example

A4 = $005002
D0 = $AAAAAAAA
MOVE.B (A4), D0



$4FFF	$12
$5000	$34
$5001	$56
$5002	$78
$5003	$9A

D0 =

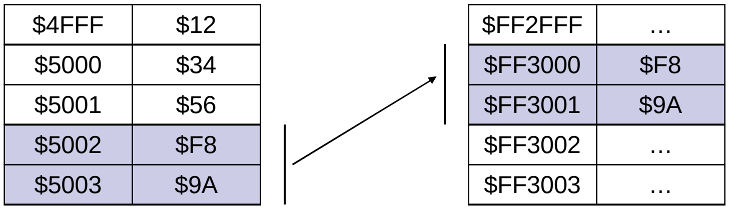
A	A	A	A	A	A	7	8

CCR
X = not effected.
N = 0 (MSB = 0)
Z = 0 (data not zero)
V = always clear.
C = always clear.

MOVE Example

A4 = $005002
A5 = $FF3000
MOVE.W (A4), (A5)

CCR
X = not effected.
N = 1 (MSB = 1)
Z = 0 (data not zero)
V = always clear.
C = always clear.

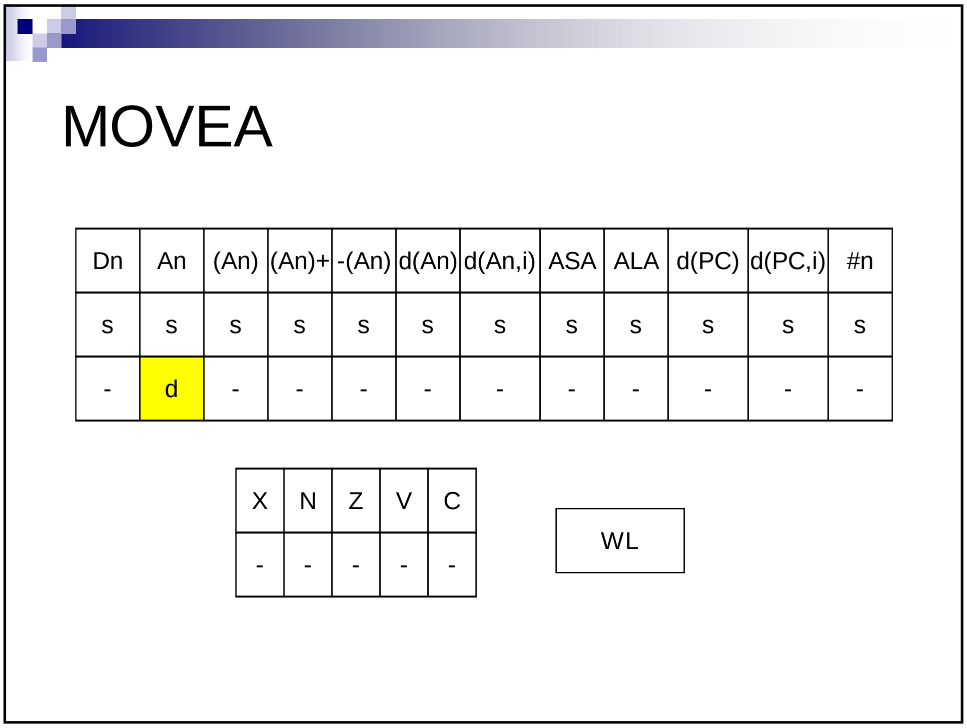


$4FFF	$12
$5000	$34
$5001	$56
$5002	$F8
$5003	$9A

$FF2FFF	…
$FF3000	$F8
$FF3001	$9A
$FF3002	…
$FF3003	…

MOVEA (Move Address)

- Used to transfer data to address register.
- Only WL can be used.
- If W used, will be sign-extended to 32-bits.
- Doesn't effect CCR.

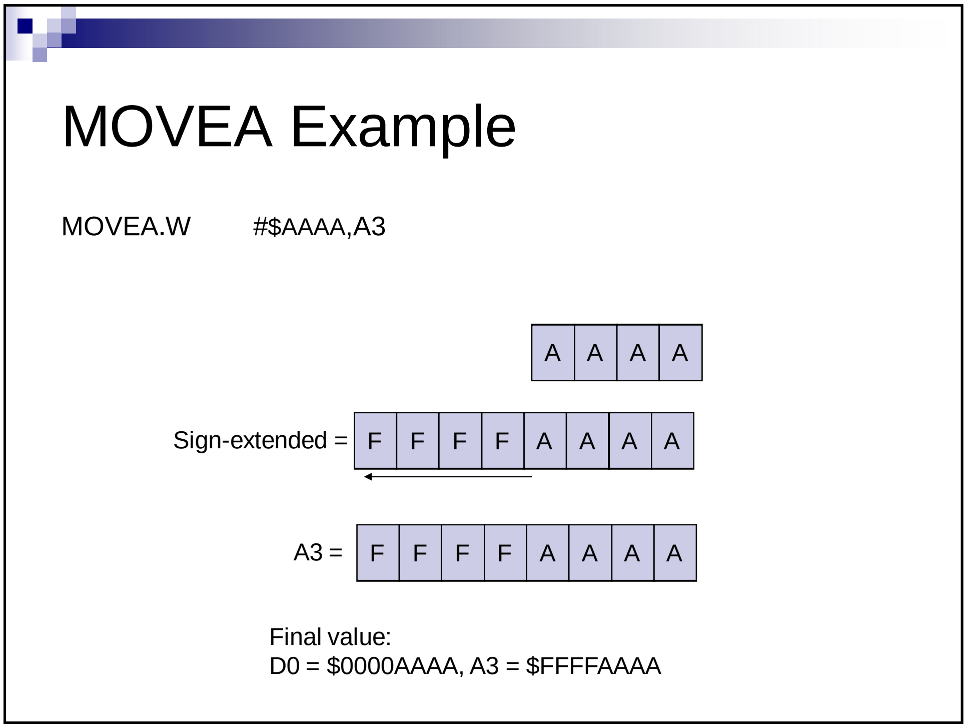


MOVEA

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	s	s	s	s	s	s	s	s	s	s	s
-	d	-	-	-	-	-	-	-	-	-	-

X	N	Z	V	C
-	-	-	-	-

WL



MOVEA Example

MOVEA.W #$AAAA,A3

					A	A	A	A
Sign-extended =	F	F	F	F	A	A	A	A
A3 =	F	F	F	F	A	A	A	A

Final value:
D0 = $0000AAAA, A3 = $FFFFAAAA

MOVEA Example

D0 = $0000AAAA
MOVEA.L D0,A3

D0 =	0	0	0	0	A	A	A	A

*L is specified, value not sign-extended.

A3 =	0	0	0	0	A	A	A	A

Final value:
D0 = $0000AAAA, A3 = $0000AAAA

MOVEQ (Move Quick)

- Moves 8-bit immediate data to data register.
- Data sign-extended to 32-bits before transfer.
- Generates smaller machine code, executes faster.

MOVEQ

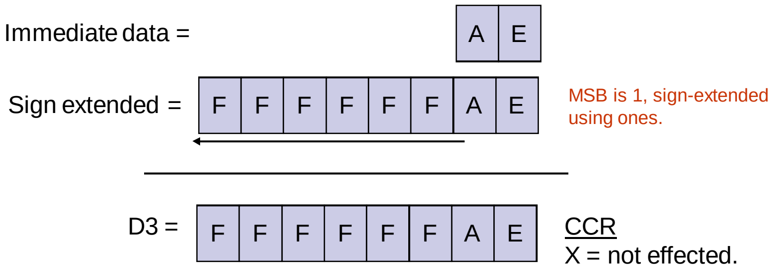
Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
-	-	-	-	-	-	-	-	-	-	-	s
d	-	-	-	-	-	-	-	-	-	-	-

X	N	Z	V	C
-	*	*	0	0

L

MOVEQ Example

MOVEQ #$AE,D3



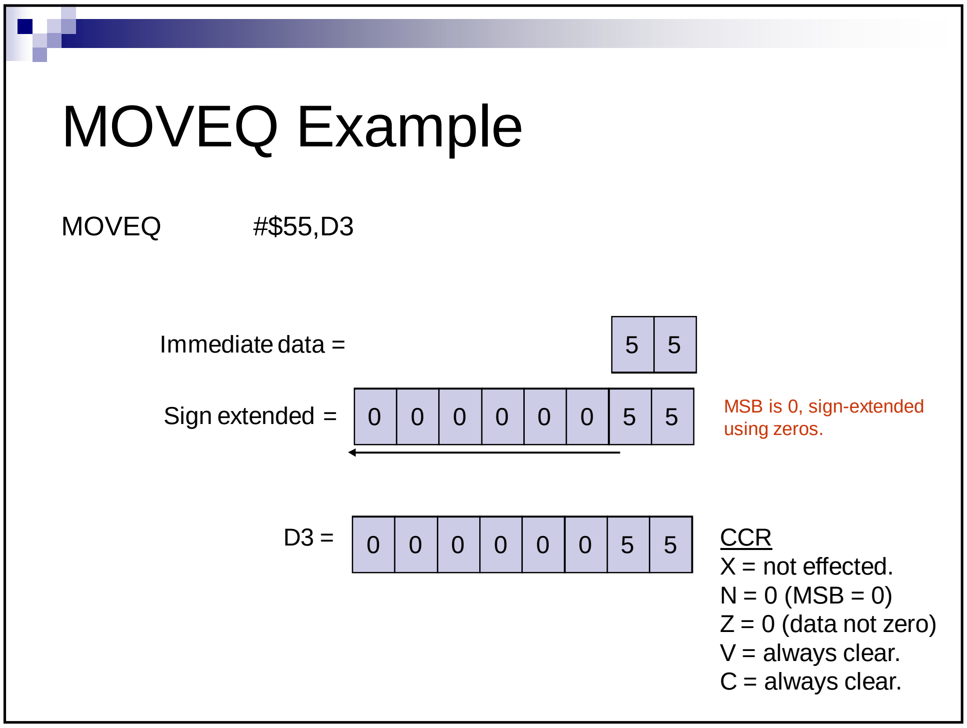
Immediate data = A E

Sign extended = F F F F F F A E

MSB is 1, sign-extended using ones.

D3 = F F F F F F A E

CCR
X = not effected.
N = 1 (MSB = 1)
Z = 0 (data not zero)
V = always clear.
C = always clear.



MOVEQ Example

MOVEQ #$55,D3

Immediate data =

5	5

Sign extended =

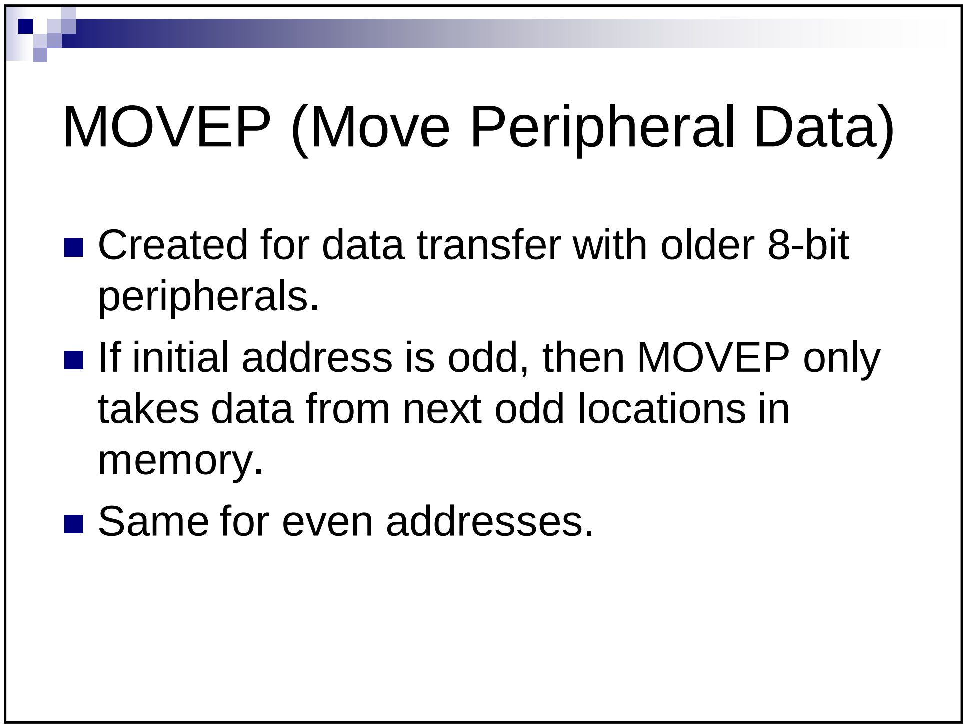
0	0	0	0	0	0	5	5

MSB is 0, sign-extended using zeros.

D3 =

0	0	0	0	0	0	5	5

CCR
X = not effected.
N = 0 (MSB = 0)
Z = 0 (data not zero)
V = always clear.
C = always clear.



MOVEP (Move Peripheral Data)

- Created for data transfer with older 8-bit peripherals.
- If initial address is odd, then MOVEP only takes data from next odd locations in memory.
- Same for even addresses.

MOVEP

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	-	-	-	-	s	-	-	-	-	-	-
d	-	-	-	-	d	-	-	-	-	-	-

X	N	Z	V	C
-	-	-	-	-

WL

MOVE vs. MOVEP

A0 = $1000
D0 = $12345678
MOVE.L D0,0(A0)

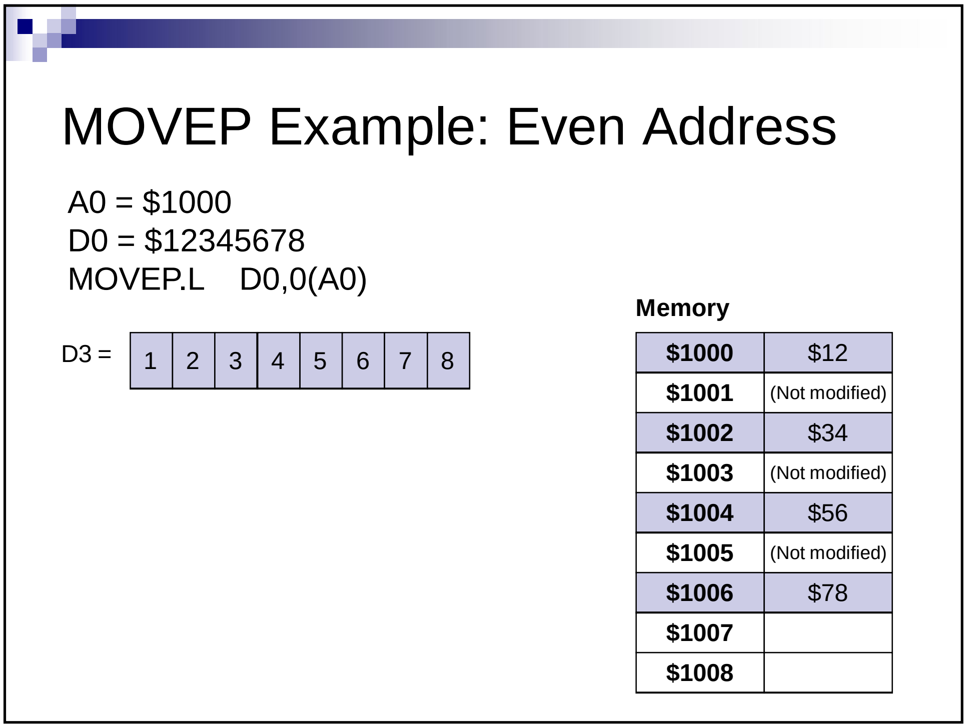
Memory

$1000	$12
$1001	$34
$1002	$56
$1003	$78
$1004	
$1005	
$1006	
$1007	
$1008	

A0 = $1000
D0 = $12345678
MOVEP.L D0,0(A0)

Memory

$1000	$12
$1001	(Not modified)
$1002	$34
$1003	(Not modified)
$1004	$56
$1005	(Not modified)
$1006	$78
$1007	
$1008	



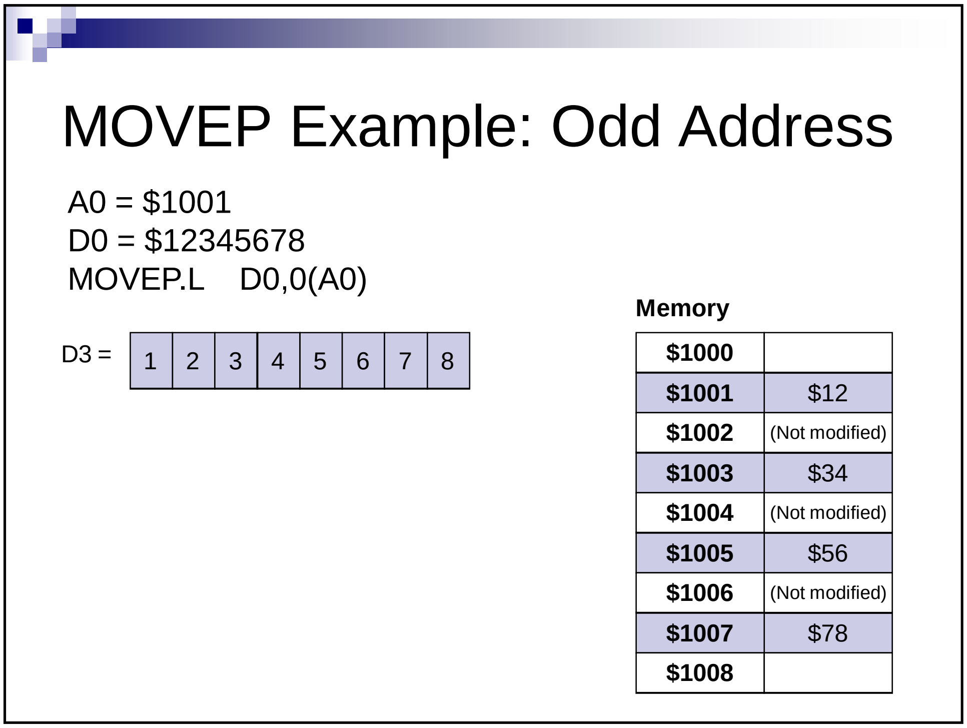
MOVEP Example: Even Address

A0 = $1000
D0 = $12345678
MOVEP.L D0,0(A0)

D3 =								
	1	2	3	4	5	6	7	8

Memory

$1000	$12
$1001	(Not modified)
$1002	$34
$1003	(Not modified)
$1004	$56
$1005	(Not modified)
$1006	$78
$1007	
$1008	



MOVEP Example: Odd Address

A0 = $1001
D0 = $12345678
MOVEP.L D0,0(A0)

D3 =								
	1	2	3	4	5	6	7	8

Memory

$1000	
$1001	$12
$1002	(Not modified)
$1003	$34
$1004	(Not modified)
$1005	$56
$1006	(Not modified)
$1007	$78
$1008	

MOVEP Example: Move Data to Register

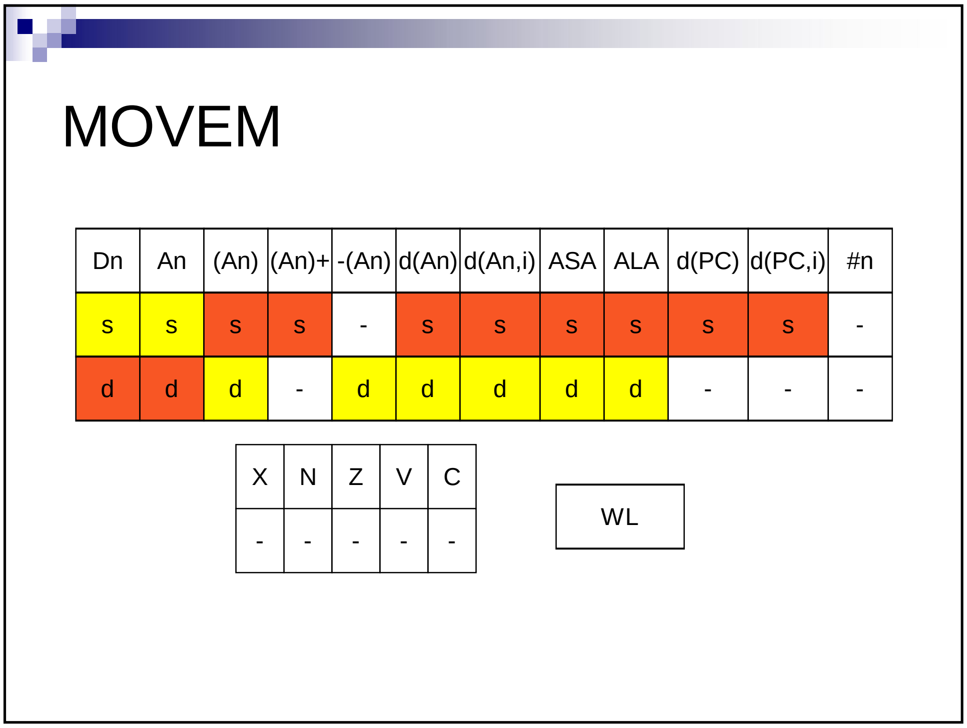
A0 = $1001
MOVEP.L 0(A0),D3

$1000	
$1001	$00
$1002	(Not taken)
$1003	$AA
$1004	(Not taken)
$1005	$BB
$1006	(Not taken)
$1007	$CC
$1008	

D3 =	0	0	A	A	B	B	C	C

MOVEM (Move Multiple Registers)

- Two uses:
  - □ Move many registers to memory at once.
  - □ Move memory locations to registers.
- Useful to save many registers to stack.
- Can use WL:
  - □ If W specified, is sign-extended.
- Writes A7-A0, then D7-D0.
- Takes back D0-D7, then A0-A7.

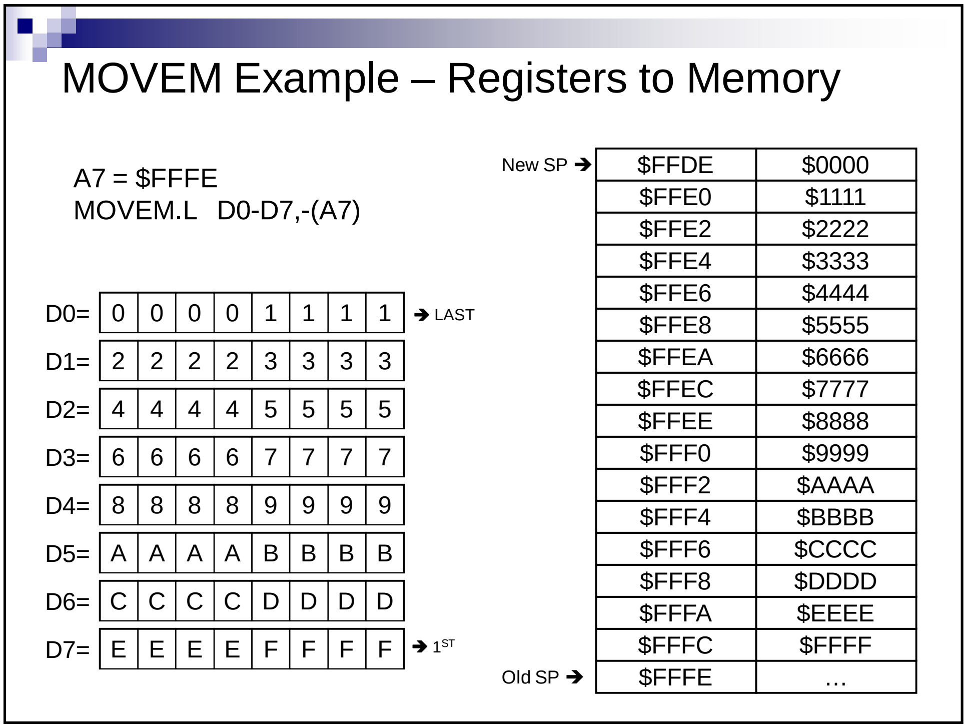


MOVEM

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	s	s	s	-	s	s	s	s	s	s	-
d	d	d	-	d	d	d	d	d	-	-	-

X	N	Z	V	C
-	-	-	-	-

WL



MOVEM Example – Registers to Memory

A7 = $FFFE
MOVEM.L   D0-D7,-(A7)

D0=	0	0	0	0	1	1	1	1	→ LAST
D1=	2	2	2	2	3	3	3	3	
D2=	4	4	4	4	5	5	5	5	
D3=	6	6	6	6	7	7	7	7	
D4=	8	8	8	8	9	9	9	9	
D5=	A	A	A	A	B	B	B	B	
D6=	C	C	C	C	D	D	D	D	
D7=	E	E	E	E	F	F	F	F	→ 1ST

New SP →	$FFDE	$0000
	$FFE0	$1111
	$FFE2	$2222
	$FFE4	$3333
	$FFE6	$4444
	$FFE8	$5555
	$FFEA	$6666
	$FFEC	$7777
	$FFEE	$8888
	$FFF0	$9999
	$FFF2	$AAAA
	$FFF4	$BBBB
	$FFF6	$CCCC
	$FFF8	$DDDD
	$FFFA	$EEEE
	$FFFC	$FFFF
Old SP →	$FFFE	…

MOVEM Example – Mem. to Registers

Old SP ➔	$FFDE	$0000
	$FFE0	$1111
	$FFE2	$2222
	$FFE4	$3333
	$FFE6	$4444
	$FFE8	$5555
	$FFEA	$6666
	$FFEC	$7777
	$FFEE	$8888
	$FFF0	$9999
	$FFF2	$AAAA
	$FFF4	$BBBB
	$FFF6	$CCCC
	$FFF8	$DDDD
	$FFFA	$EEEE
	$FFFC	$FFFF
New SP ➔	$FFFE	…

A7 = $FFDE
MOVEM.L (A7)+, D0-D7

D0=	0	0	0	0	1	1	1	1	➔ 1st
D1=	2	2	2	2	3	3	3	3	
D2=	4	4	4	4	5	5	5	5	
D3=	6	6	6	6	7	7	7	7	
D4=	8	8	8	8	9	9	9	9	
D5=	A	A	A	A	B	B	B	B	
D6=	C	C	C	C	D	D	D	D	
D7=	E	E	E	E	F	F	F	F	➔ LAST

Saving/Recovering All Registers

- To save all registers to stack:
  - □ MOVEM.L D0-D7/A0-A7, -(A7)
  - □ Put A7 first, then A6, A5, A4,…
  - □ Then put D7, D6, D5, D4, …
- To get back all registers from stack:
  - □ MOVEM.L (A7)+, D0-D7/A0-A7
  - □ Gets D0 first, then D1, D2, …, D7.
  - □ Then, get A0, A1, …, A7.

MOVEM Example – Selected Registers to Memory

A7 = $FFFE
MOVEM.L   D0/A5/A4/D3,-(A7)

D0=	0	0	0	0	1	1	1	1	➔ LAST
D3=	2	2	2	2	3	3	3	3	
A4=	8	8	8	8	9	9	9	9	
A5=	A	A	A	A	B	B	B	B	➔ 1st

*MOVEM always puts A7-A0, then D7-D0
It doesn't matter how you arrange them.

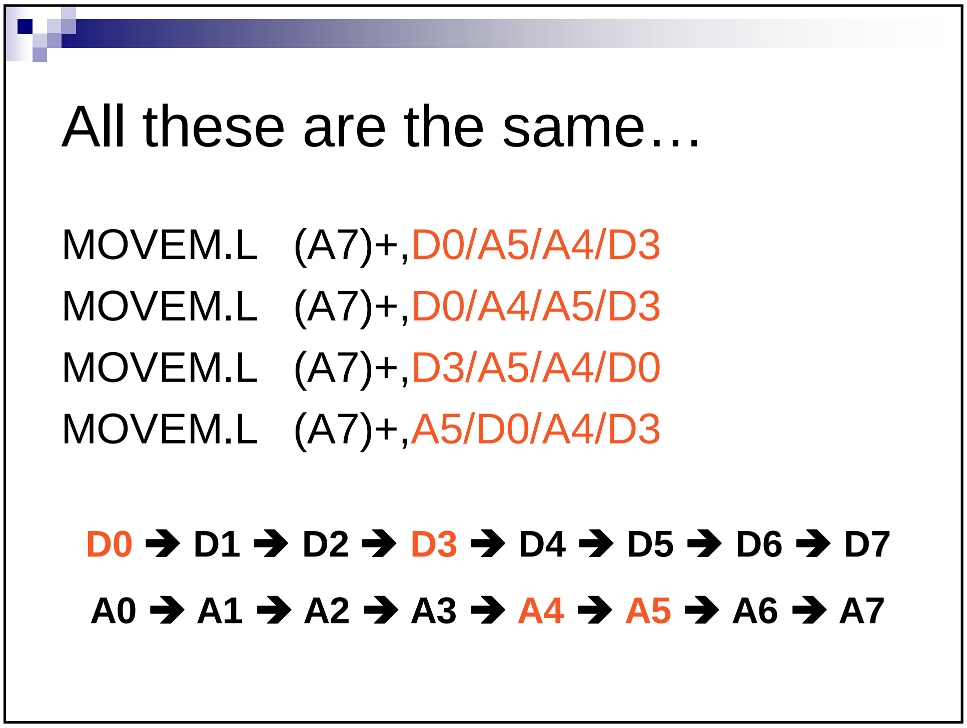
New SP ➔	$FFDE	
	$FFE0	
	$FFE2	
	$FFE4	
	$FFE6	
	$FFE8	
	$FFEA	
	$FFEC	
	$FFEE	$0000
	$FFF0	$1111
	$FFF2	$2222
	$FFF4	$3333
	$FFF6	$8888
	$FFF8	$9999
	$FFFA	$AAAA
	$FFFC	$BBBB
Old SP ➔	$FFFE	…

All these are the same…

MOVEM.L   D0/A5/A4/D3,-(A7)
MOVEM.L   D0/A4/A5/D3,-(A7)
MOVEM.L   D3/A5/A4/D0,-(A7)
MOVEM.L   A5/D0/A4/D3,-(A7)

A7 ➔ A6 ➔ A5 ➔ A4 ➔ A3 ➔ A2 ➔ A1 ➔ A0

D7 ➔ D6 ➔ D5 ➔ D4 ➔ D3 ➔ D2 ➔ D1 ➔ D0



All these are the same…

```
MOVEM.L   (A7)+,D0/A5/A4/D3
MOVEM.L   (A7)+,D0/A4/A5/D3
MOVEM.L   (A7)+,D3/A5/A4/D0
MOVEM.L   (A7)+,A5/D0/A4/D3
```

D0 → D1 → D2 → D3 → D4 → D5 → D6 → D7

A0 → A1 → A2 → A3 → A4 → A5 → A6 → A7

Exchange & Swap

EXG (Exchange Registers)

- Exchanges contents of registers.
- Can only use L.
- Doesn’t effect CCR.

EXG

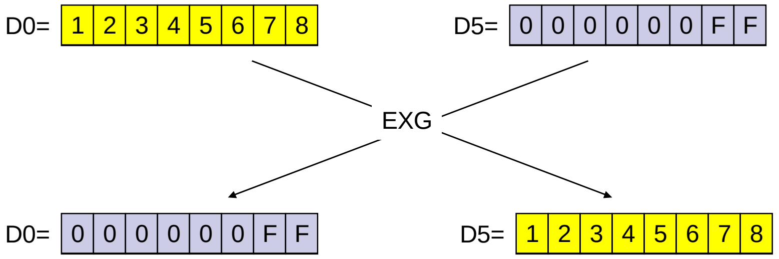
Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	s	-	-	-	-	-	-	-	-	-	-
d	d	-	-	-	-	-	-	-	-	-	-

X	N	Z	V	C
-	-	-	-	-

L

EXG Example

D0 = $12345678
D5 = $000000FF
EXG   D0,D5



D0=	1	2	3	4	5	6	7	8

D5=	0	0	0	0	0	0	F	F

D0=	0	0	0	0	0	0	F	F

D5=	1	2	3	4	5	6	7	8

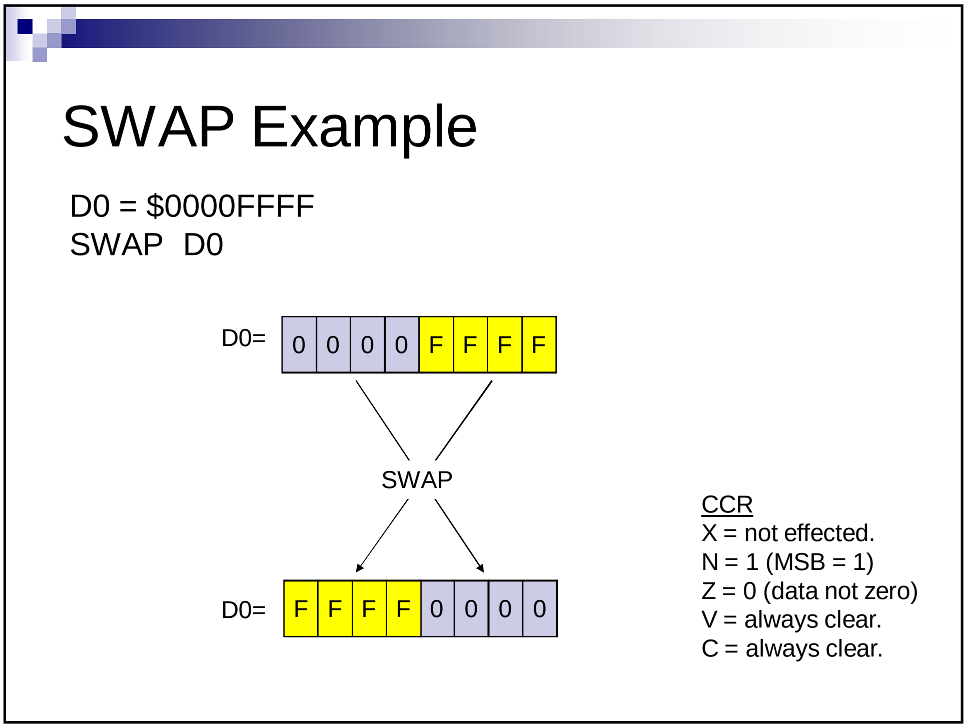
SWAP (Swap Register Halves)

- Swaps contents of upper and lower words in register.

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
s	-	-	-	-	-	-	-	-	-	-	-
d	-	-	-	-	-	-	-	-	-	-	-

X	N	Z	V	C
-	*	*	0	0

L



SWAP Example

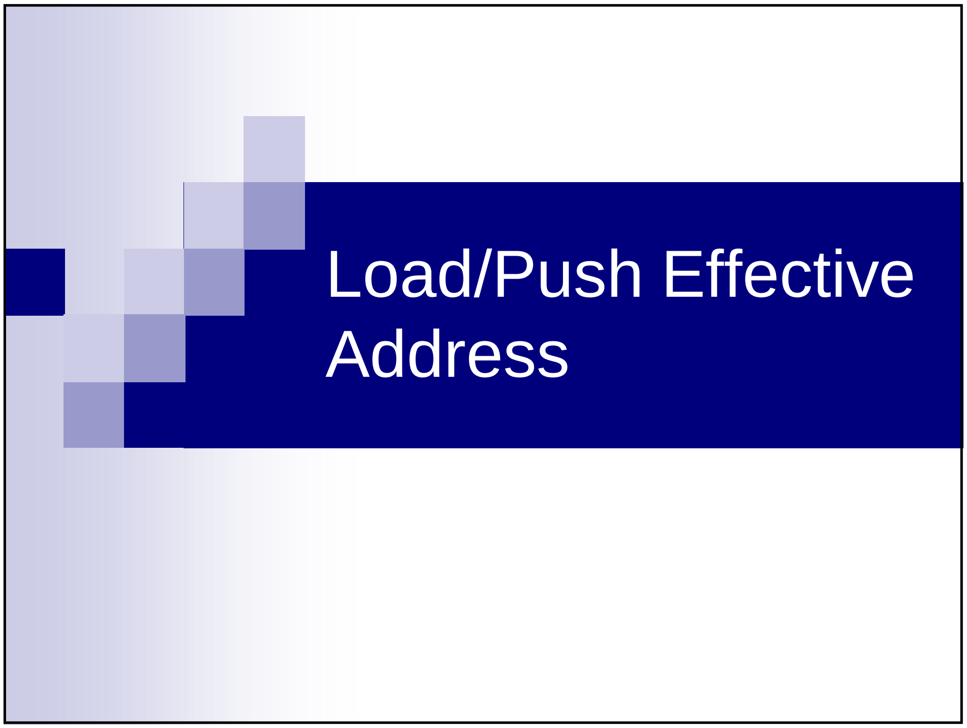
D0 = $0000FFFF
SWAP D0

D0=	0	0	0	0	F	F	F	F

SWAP

D0=	F	F	F	F	0	0	0	0

CCR
X = not effected.
N = 1 (MSB = 1)
Z = 0 (data not zero)
V = always clear.
C = always clear.



Load/Push Effective Address

LEA (Load Effective Address)

- Loads effective address into address register.

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
-	s	s	-	-	s	s	s	s	s	s	-
-	d	-	-	-	-	-	-	-	-	-	-

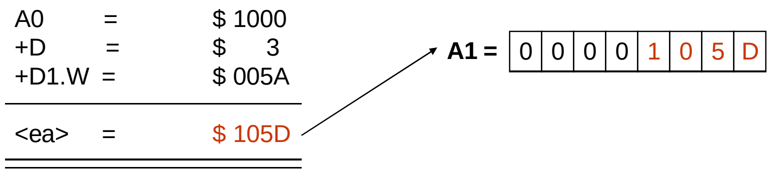
X	N	Z	V	C
-	-	-	-	-

L

LEA Example

- Find out the effective address of 3(A0,D1.W) when A0 = $1000, and D1 = $005A.

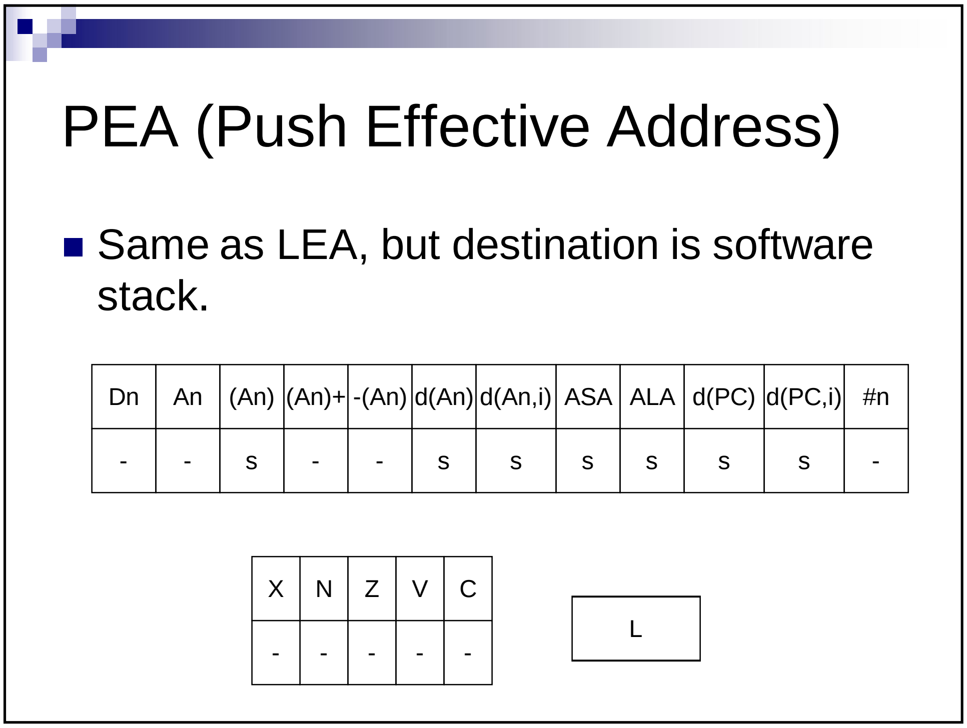
```
LEA     3(A0,D1.W),A1
```



```
A0      =     $ 1000
+D      =     $    3
+D1.W   =     $ 005A
--------------------
<ea>    =     $ 105D
====================
```

A1 =

0	0	0	0	1	0	5	D



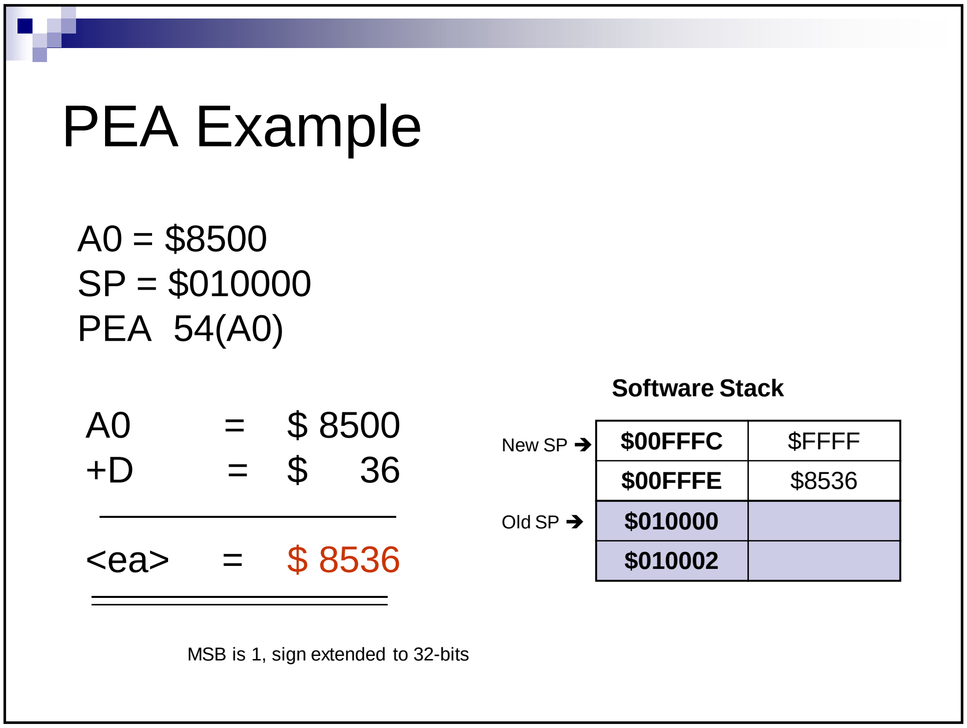
PEA (Push Effective Address)

- Same as LEA, but destination is software stack.

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,i)	ASA	ALA	d(PC)	d(PC,i)	#n
-	-	s	-	-	s	s	s	s	s	s	-

X	N	Z	V	C
-	-	-	-	-

L



PEA Example

A0 = $8500
SP = $010000
PEA 54(A0)

A0 = $ 8500
+D = $ 36

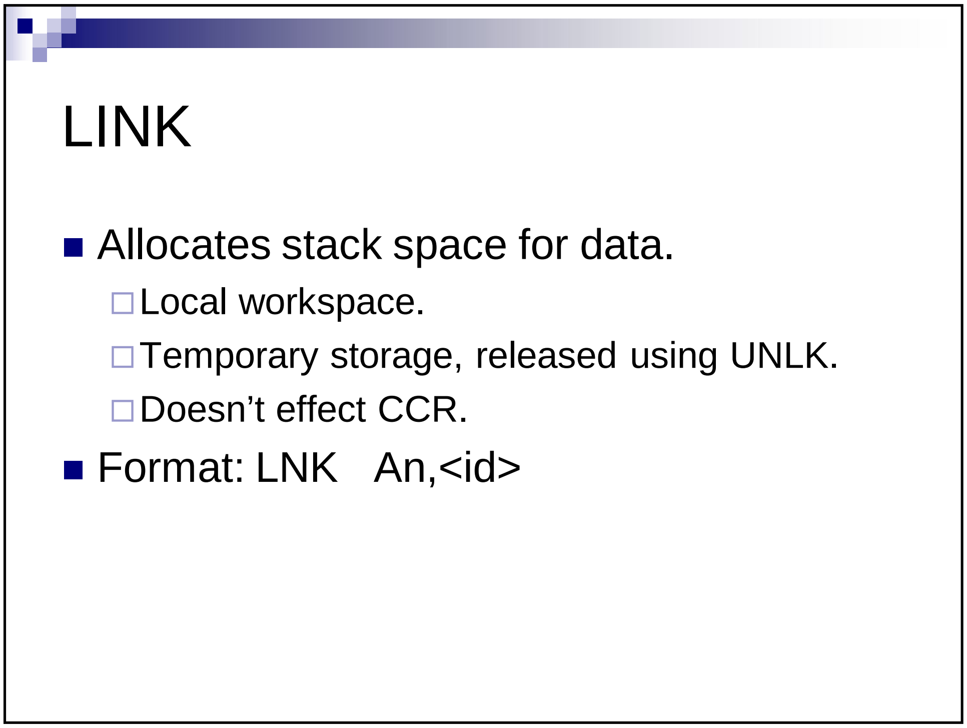
<ea> = $ 8536

MSB is 1, sign extended to 32-bits

Software Stack

New SP →	**$00FFFC**	$FFFF
	$00FFFE	$8536
Old SP →	**$010000**	
	$010002	

Link & Unlink Stack Pointer*



LINK

- Allocates stack space for data.
  - □ Local workspace.
  - □ Temporary storage, released using UNLK.
  - □ Doesn’t effect CCR.
- Format: LNK   An,<id>

How LINK works

- An value stored into SP.
- SP stored into An.
- SP modified using immediate data value:
  - □ Immediate data must be negative.

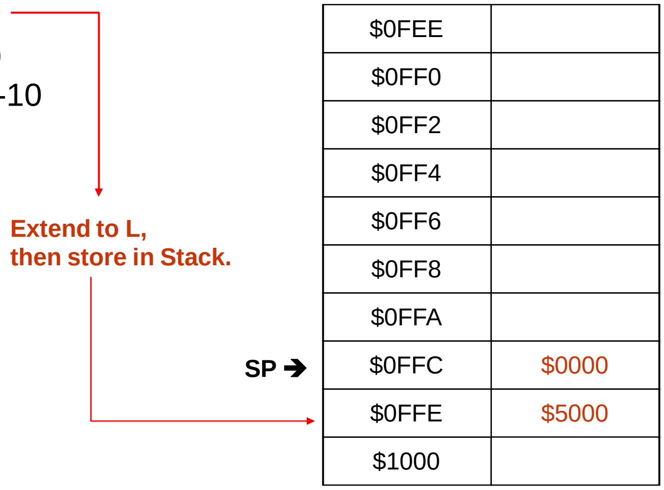
How LINK Works – Before Execution

A0 = $5000
SP = $1000
LINK   A0,#-10

	$0FEE	
	$0FF0	
	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	
	$0FFC	
	$0FFE	
SP ➔	$1000	

How LINK Works – Store An into Stack

A0 = $5000
SP = $1000
LINK   A0,#-10

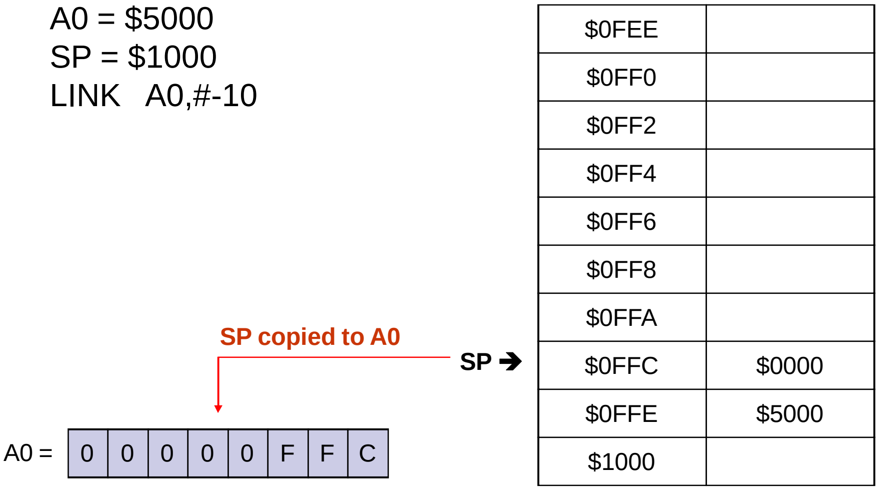


Extend to L,
then store in Stack.

$0FEE	
$0FF0	
$0FF2	
$0FF4	
$0FF6	
$0FF8	
$0FFA	
SP ➔ $0FFC	$0000
$0FFE	$5000
$1000	

How LINK Works – Copy Stack Pointer to An

A0 = $5000
SP = $1000
LINK   A0,#-10

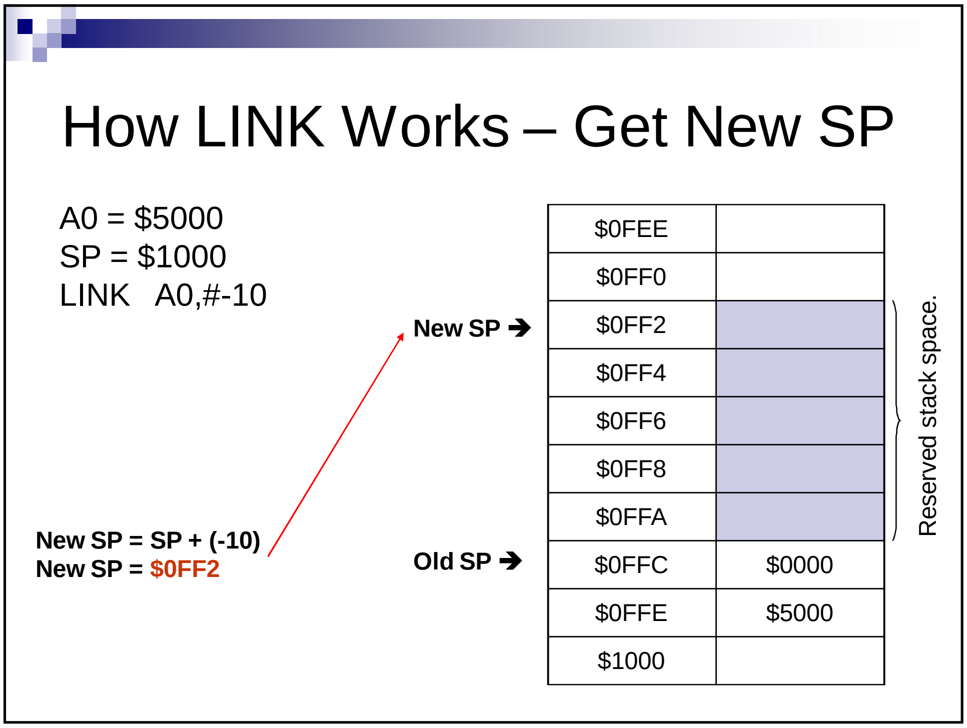


SP copied to A0

$0FEE	
$0FF0	
$0FF2	
$0FF4	
$0FF6	
$0FF8	
$0FFA	
SP ➔ $0FFC	$0000
$0FFE	$5000
$1000	

A0 =

0	0	0	0	0	F	F	C



How LINK Works – Get New SP

A0 = $5000
SP = $1000
LINK A0,#-10

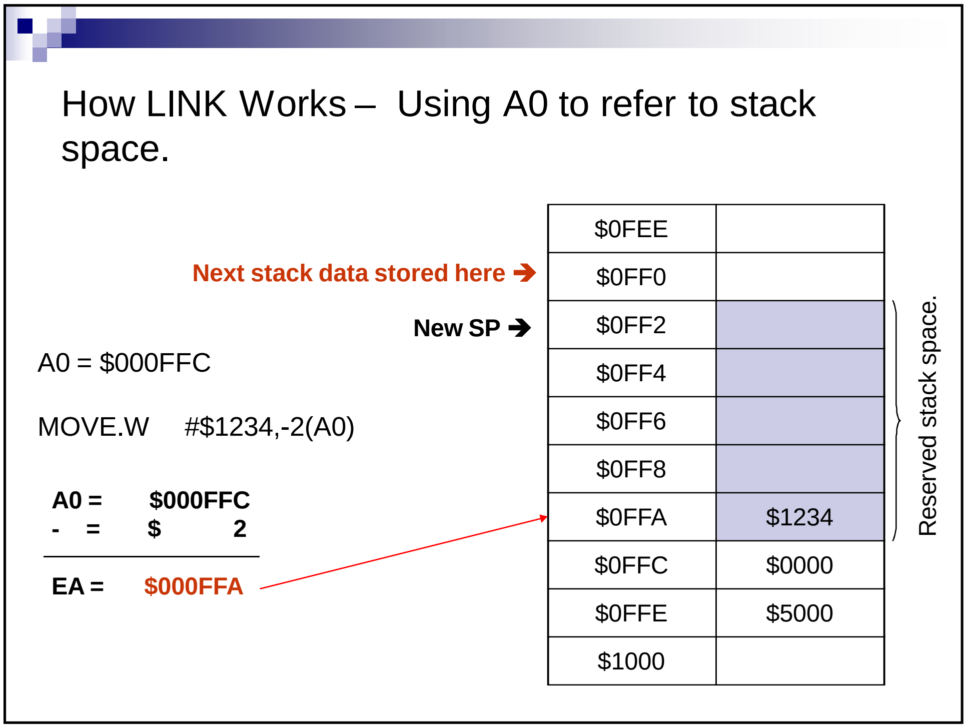
New SP = SP + (-10)
New SP = $0FF2

New SP ➔ $0FF2

Old SP ➔ $0FFC

$0FEE	
$0FF0	
$0FF2	
$0FF4	
$0FF6	
$0FF8	
$0FFA	
$0FFC	$0000
$0FFE	$5000
$1000	

Reserved stack space.



How LINK Works – Using A0 to refer to stack space.

Next stack data stored here ➔ $0FF0

New SP ➔ $0FF2

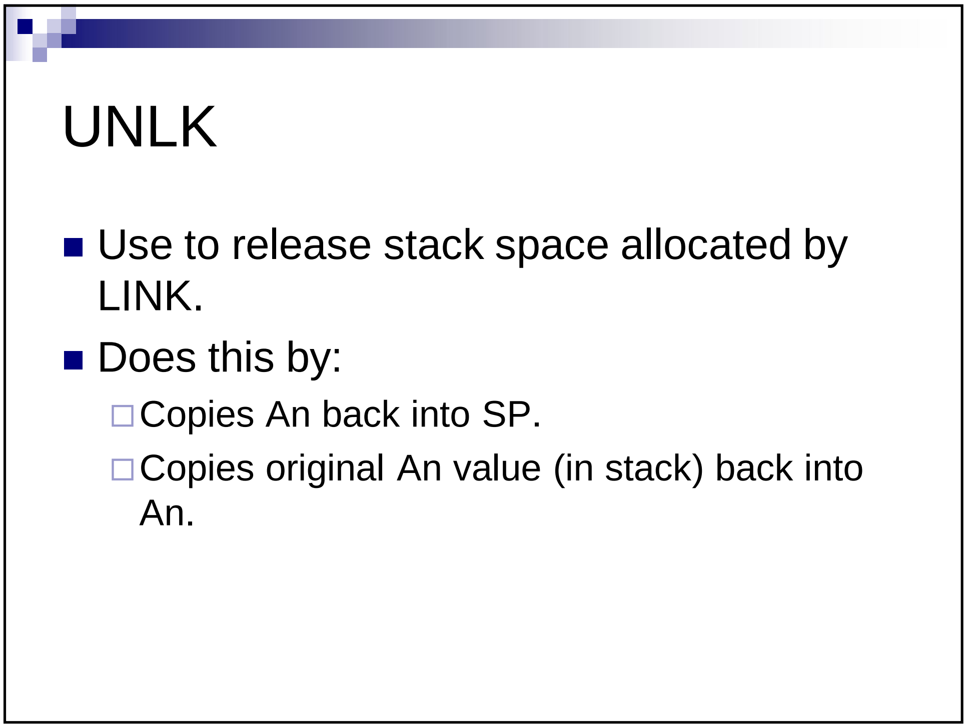
A0 = $000FFC

MOVE.W #$1234,-2(A0)

A0 =	$000FFC
- =	$ 2
EA =	$000FFA

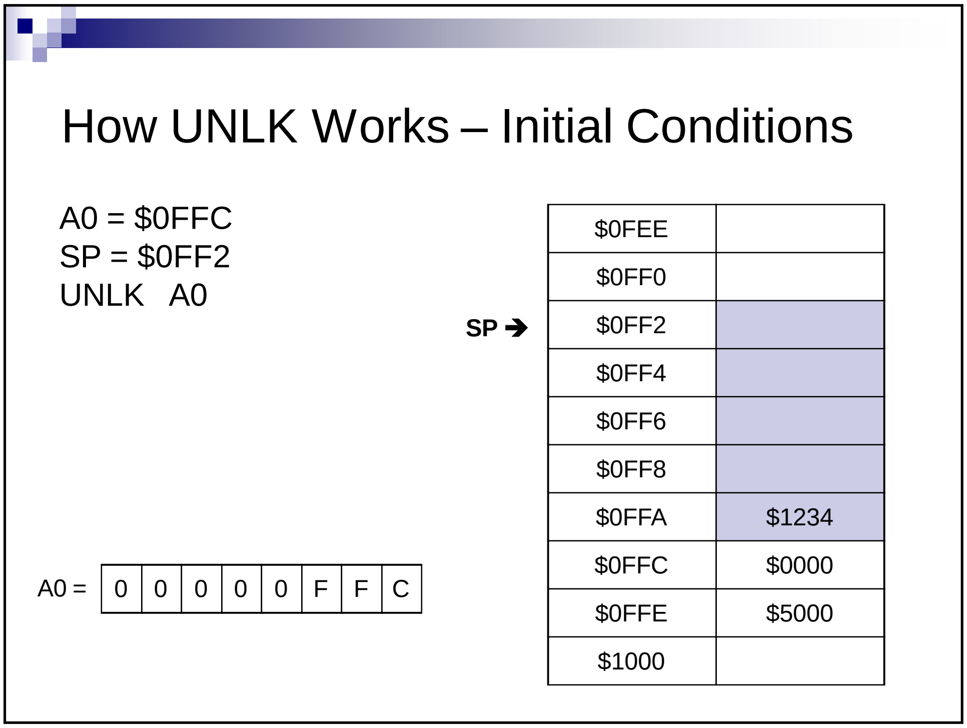
$0FEE	
$0FF0	
$0FF2	
$0FF4	
$0FF6	
$0FF8	
$0FFA	$1234
$0FFC	$0000
$0FFE	$5000
$1000	

Reserved stack space.



UNLK

- Use to release stack space allocated by LINK.
- Does this by:
  - □ Copies An back into SP.
  - □ Copies original An value (in stack) back into An.

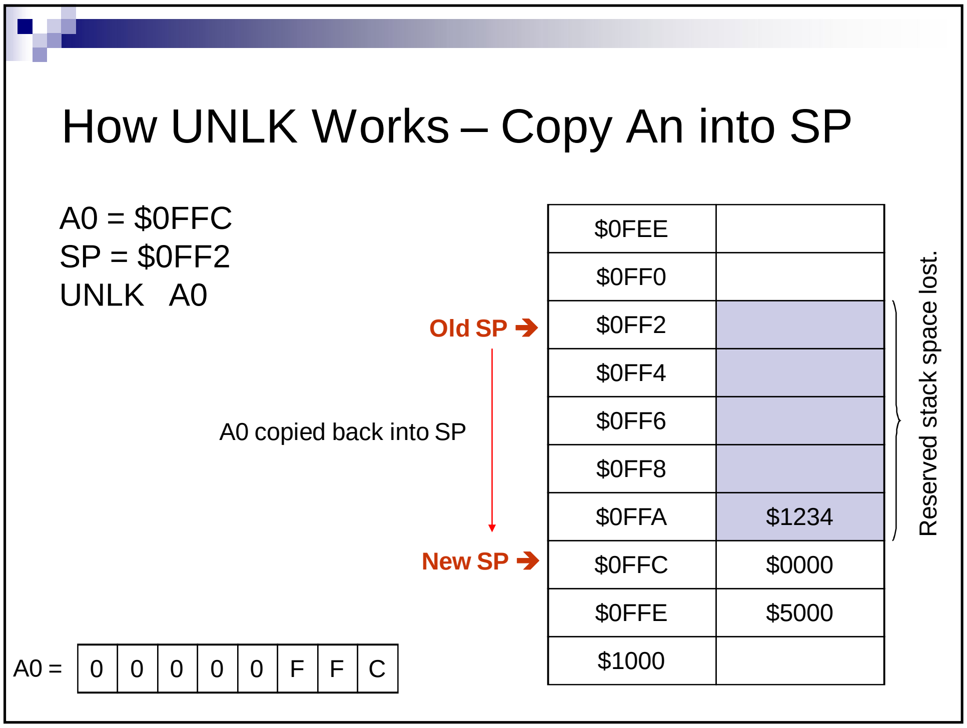


How UNLK Works – Initial Conditions

A0 = $0FFC
SP = $0FF2
UNLK A0

	$0FEE	
	$0FF0	
SP ➔	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	$1234
	$0FFC	$0000
	$0FFE	$5000
	$1000	

A0 =	0	0	0	0	0	F	F	C



How UNLK Works – Copy An into SP

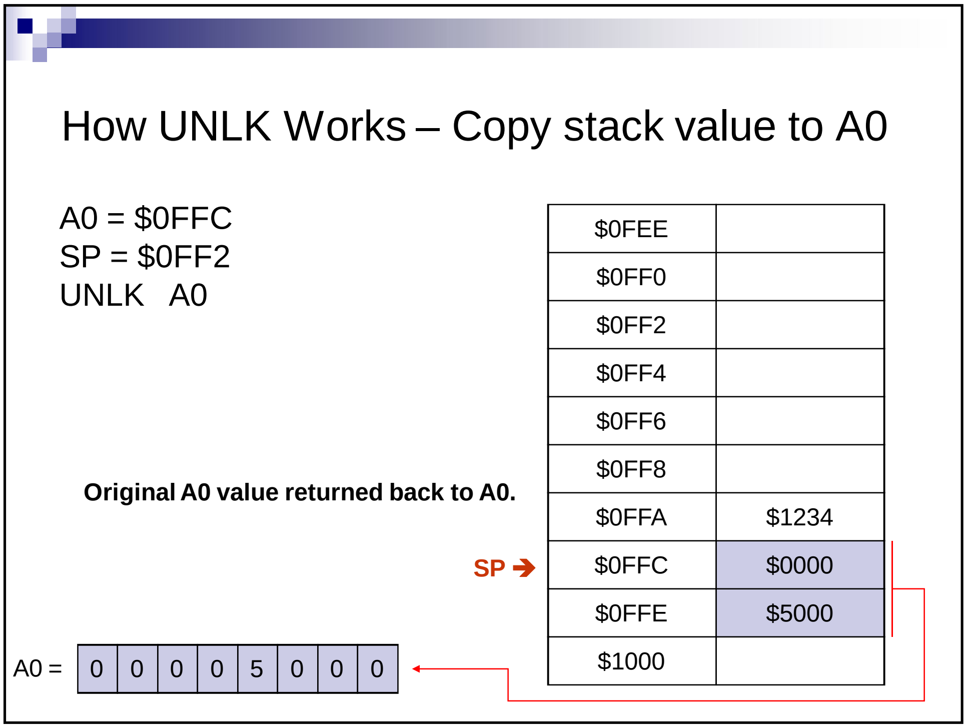
A0 = $0FFC
SP = $0FF2
UNLK   A0

A0 copied back into SP

	$0FEE	
	$0FF0	
Old SP →	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	$1234
New SP →	$0FFC	$0000
	$0FFE	$5000
	$1000	

Reserved stack space lost.

A0 =	0	0	0	0	0	F	F	C



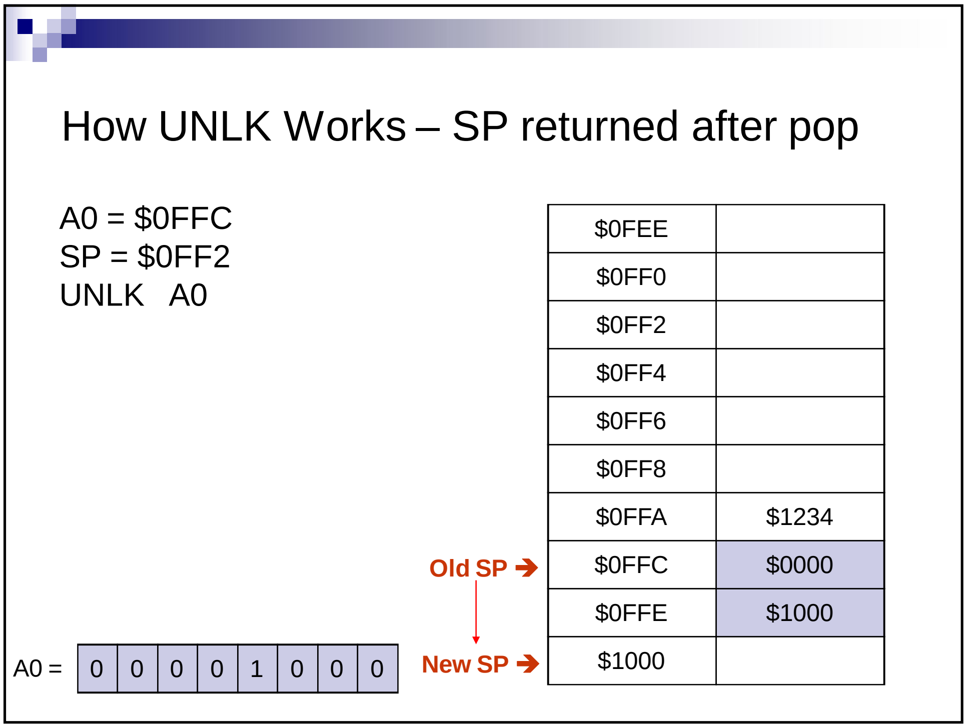
How UNLK Works – Copy stack value to A0

A0 = $0FFC
SP = $0FF2
UNLK   A0

Original A0 value returned back to A0.

	$0FEE	
	$0FF0	
	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	$1234
SP →	$0FFC	$0000
	$0FFE	$5000
	$1000	

A0 =	0	0	0	0	5	0	0	0

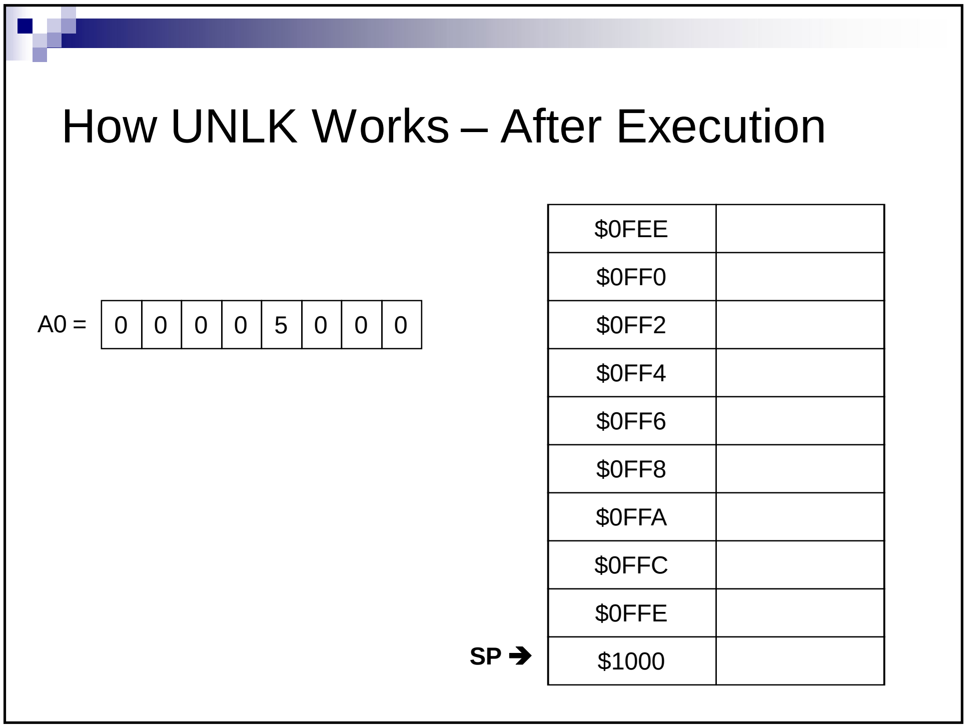


How UNLK Works – SP returned after pop

A0 = $0FFC
SP = $0FF2
UNLK   A0

	$0FEE	
	$0FF0	
	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	$1234
Old SP ➔	$0FFC	$0000
	$0FFE	$1000
New SP ➔	$1000	

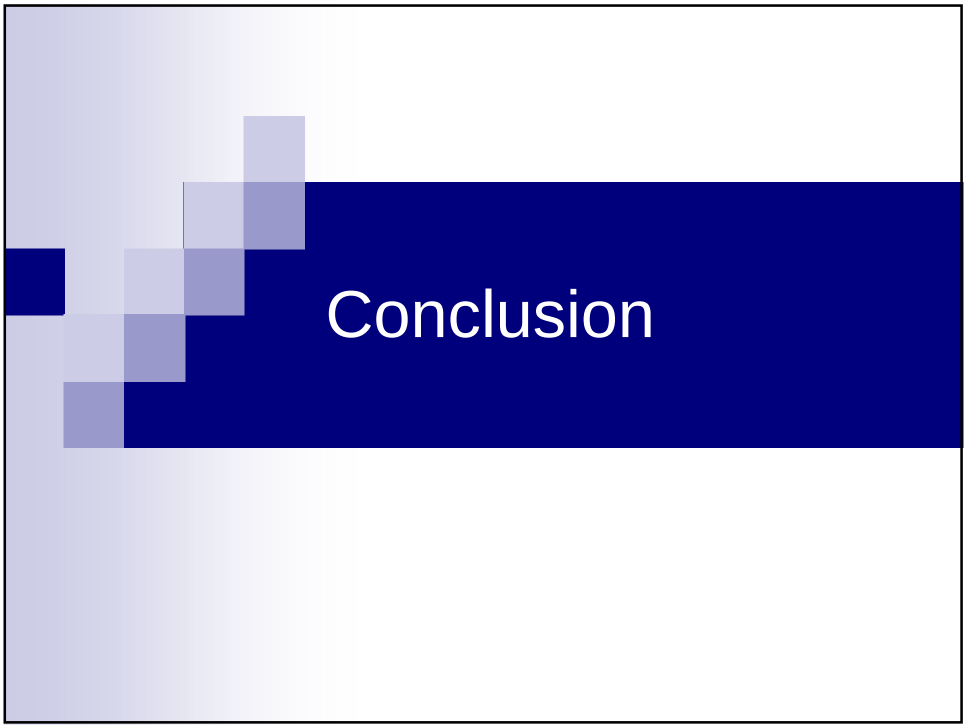
A0 =								
	0	0	0	0	1	0	0	0



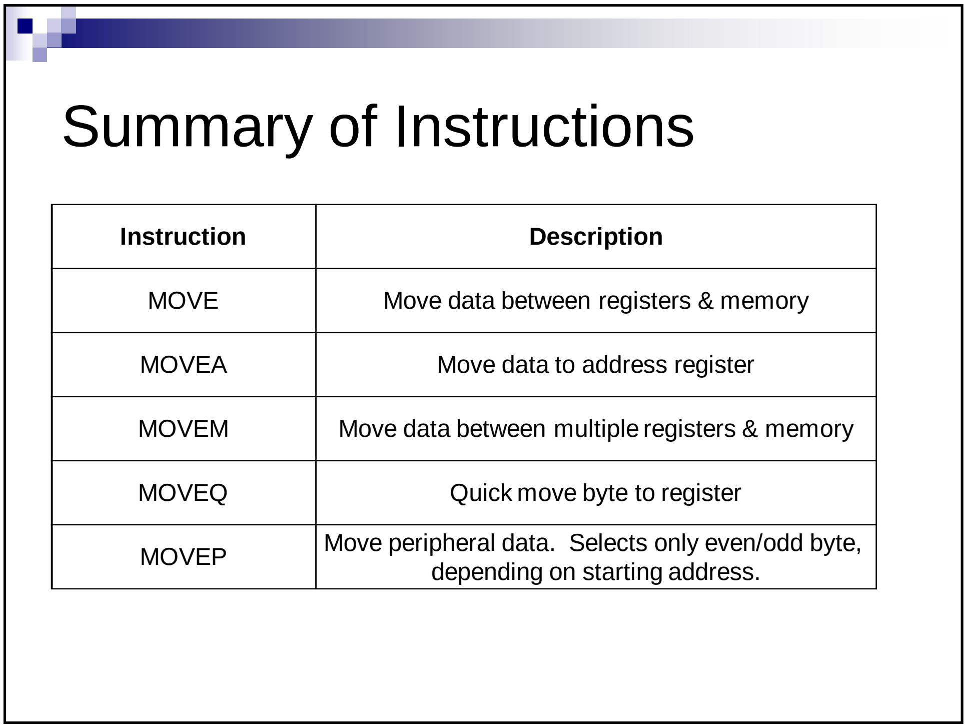
How UNLK Works – After Execution

A0 =								
	0	0	0	0	5	0	0	0

	$0FEE	
	$0FF0	
	$0FF2	
	$0FF4	
	$0FF6	
	$0FF8	
	$0FFA	
	$0FFC	
	$0FFE	
SP ➔	$1000	

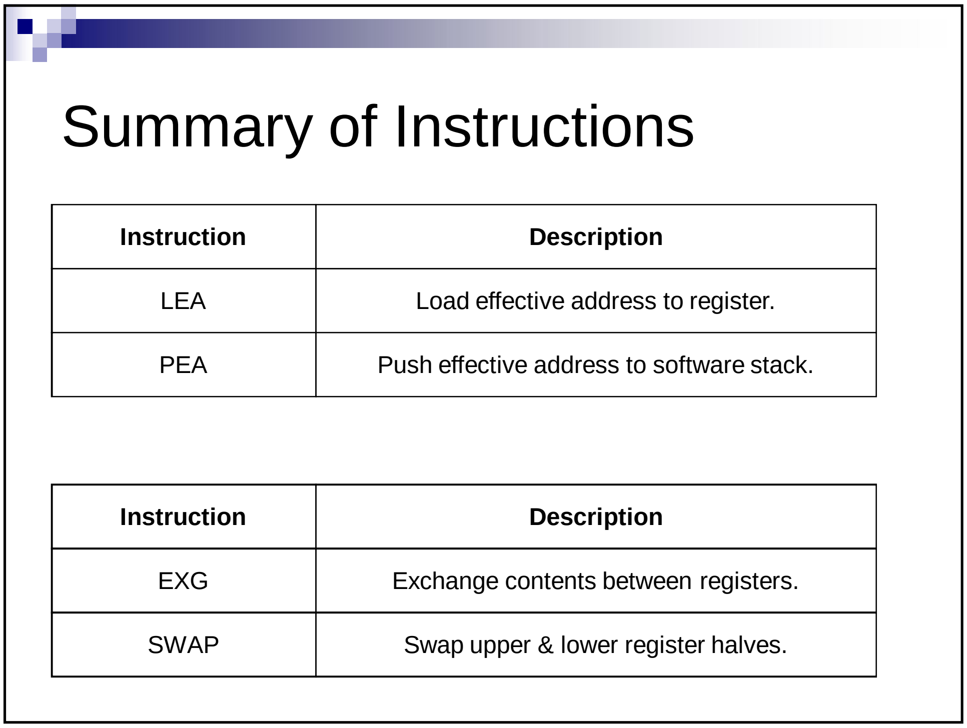


Conclusion



Summary of Instructions

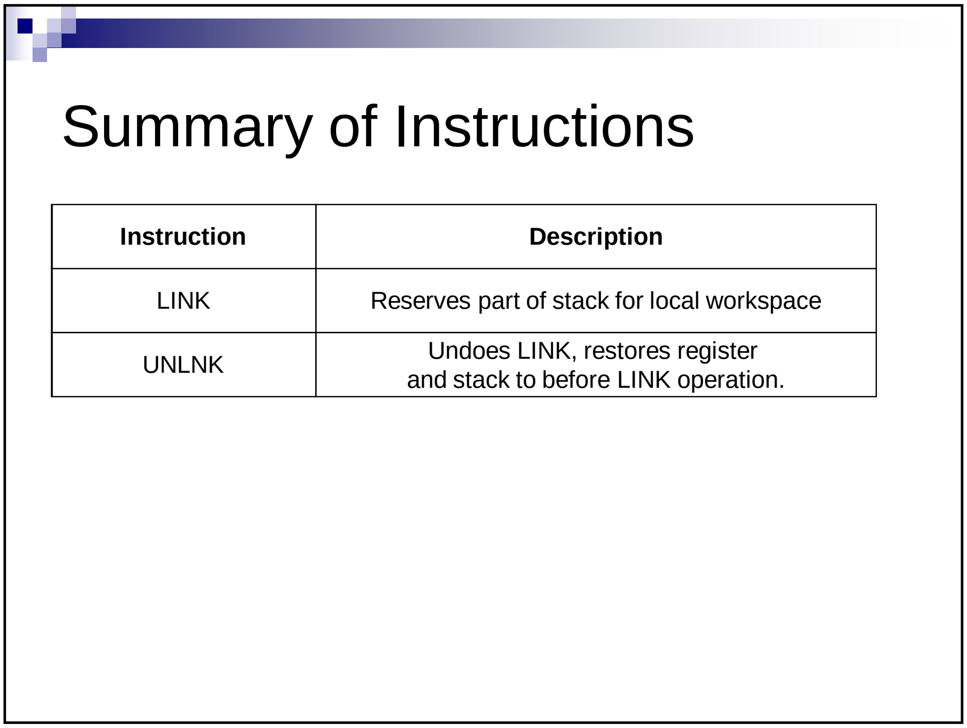
Instruction	Description
MOVE	Move data between registers & memory
MOVEA	Move data to address register
MOVEM	Move data between multiple registers & memory
MOVEQ	Quick move byte to register
MOVEP	Move peripheral data. Selects only even/odd byte, depending on starting address.



Summary of Instructions

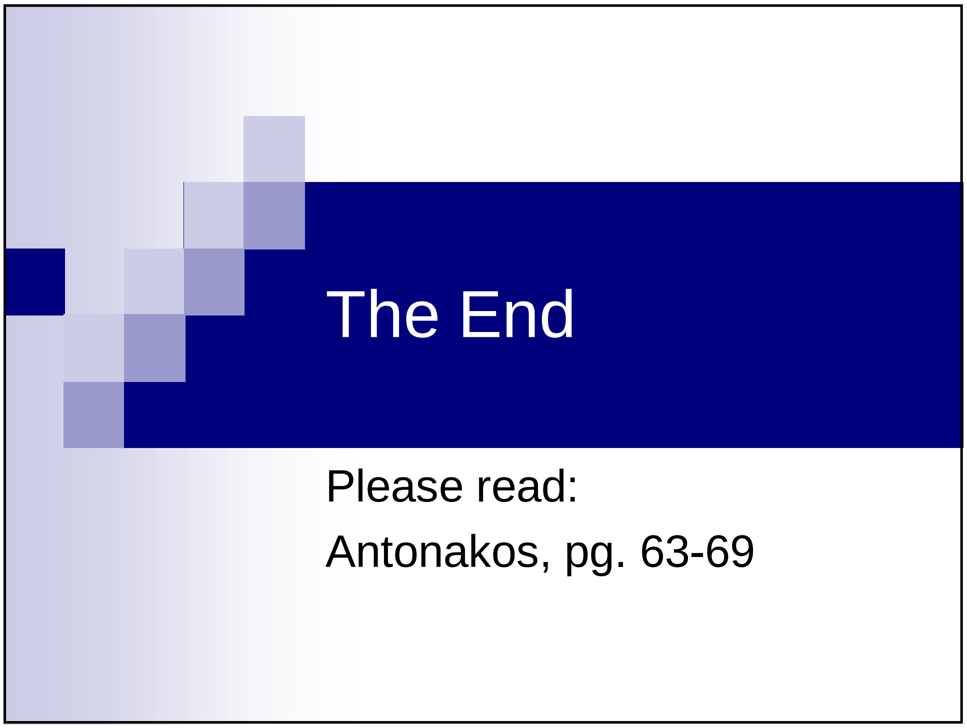
Instruction	Description
LEA	Load effective address to register.
PEA	Push effective address to software stack.

Instruction	Description
EXG	Exchange contents between registers.
SWAP	Swap upper & lower register halves.



Summary of Instructions

Instruction	Description
LINK	Reserves part of stack for local workspace
UNLNK	Undoes LINK, restores register and stack to before LINK operation.



The End

Please read:

Antonakos, pg. 63-69