SYSC3601
Microprocessor Systems

Unit 3:
The Intel 8086 Addressing Modes & Instruction Encoding

Topics/Reading

1. Addressing modes (Brey Ch 3)

2. Assembly and Machine Language

   Instruction Encoding (Brey Ch 4-7 and Appendix B)

Addressing Modes

- We will use the MOV instruction to discuss the various addressing modes.
- $\underbrace{\text{MOV}}_{\text{opcode}}\quad\underbrace{\text{Dst,Src}}_{\text{operands}}$ (i.e. Dst=Src after MOV)
- MOV transfers bytes or words of data between registers or between registers and memory.
- MOV creates a copy of the data
  - (i.e., it does not alter the source) and it does **NOT** set the flags.

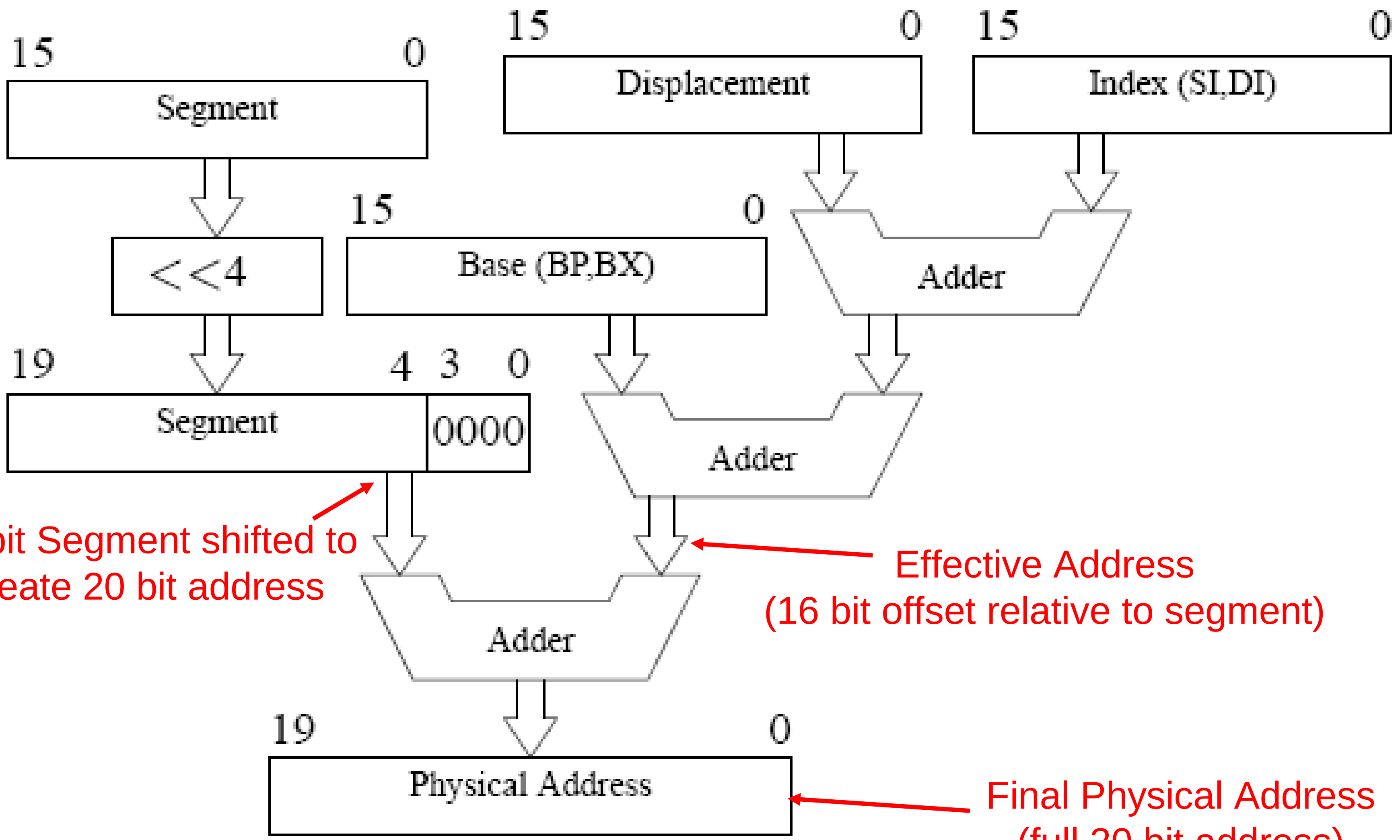
Addressing Modes

- MOV rules:
  1. Source and destination must be the same size.
  2. Segment to segment register move is not allowed (segment value would be overwritten and lost).
  3. CS register may not be changed by a MOV (`MOV CS` would clobber program flow).
  4. Memory to memory moves are not allowed, *except* as strings, eg `MOVS [DI],[SI]`

Addressing Modes - Effective Address (EA)

- The execution unit is responsible for computing the EA and passes the results to the BIU which combines it with the segment register.
- The EA is the offset that the execution unit calculates for a memory operand.
  - it is an unsigned 16 bit number that expresses the operand’s distance (in bytes) from the beginning of the segment in which it resides.
  - the EA is the sum of a displacement, contents of a base register, and contents of an index register.
  - The addressing mode determines the registers needed to compute the EA.

Addressing Modes - Effective Address (EA)

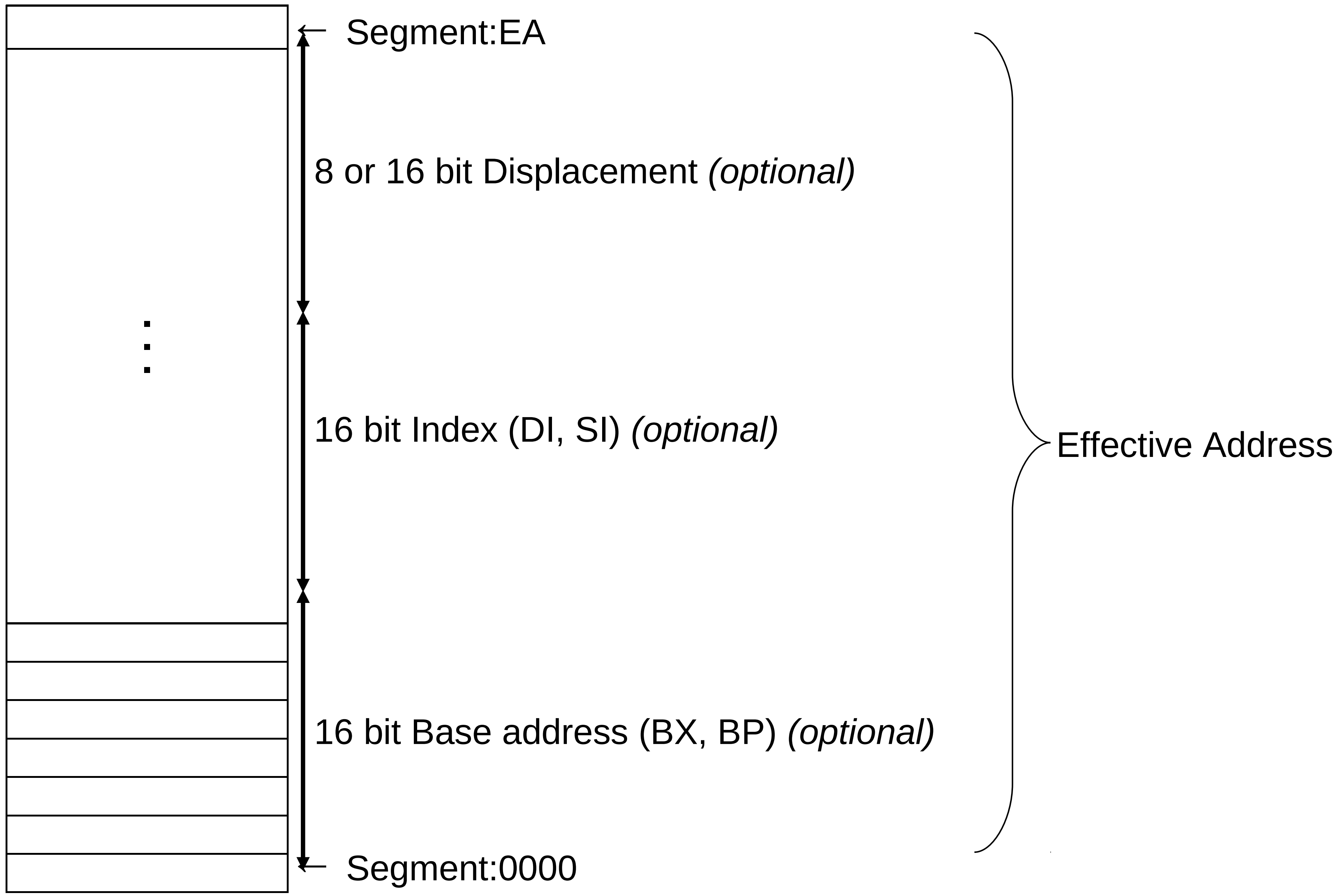


16 bit Segment shifted to create 20 bit address

Effective Address (16 bit offset relative to segment)

Final Physical Address (full 20 bit address)

Addressing Modes - Effective Address (EA)


Segment:EA
8 or 16 bit Displacement *(optional)*
16 bit Index (DI, SI) *(optional)*
Effective Address
16 bit Base address (BX, BP) *(optional)*
Segment:0000

Addressing Modes

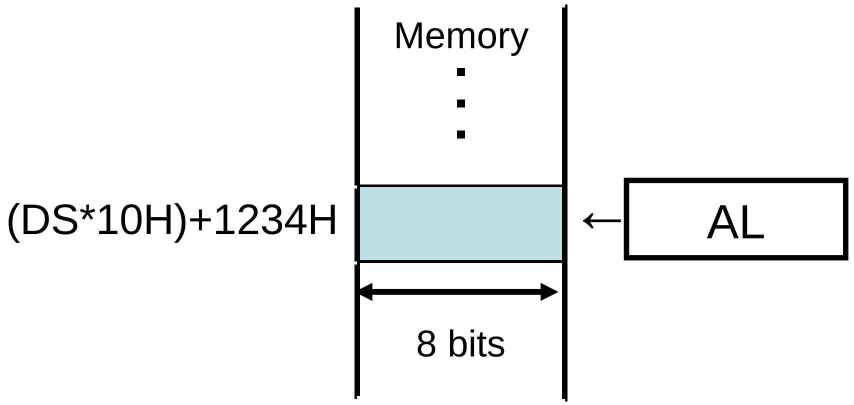
- **Register addressing**
  - Data is in the registers specified in the instructions.
  - eg: `MOV  AX,BX`

- **Immediate addressing**
  - Data is a **constant** and is part of the instruction.
  - eg: `MOV  AX,3AH`

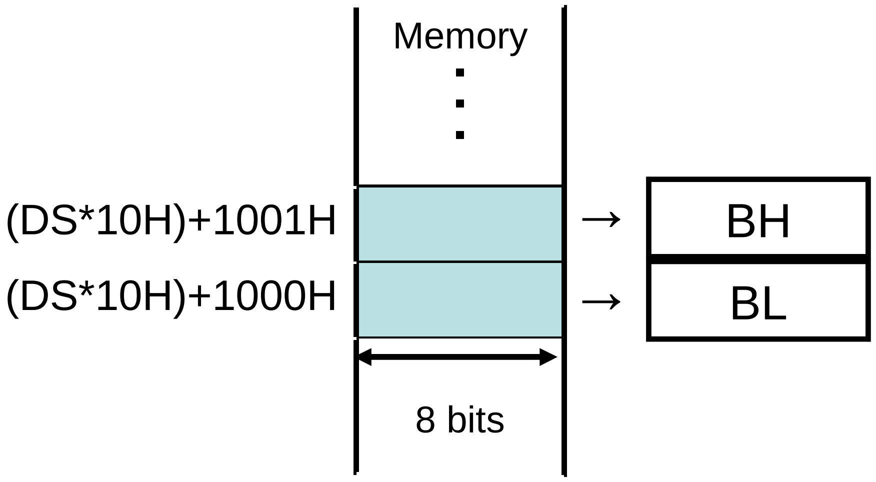
Addressing Modes

- **Direct addressing**
  - The 16 bit effective address is part of the instruction. (can think of this as 'displacement only')

ex1: MOV DS:1234H,AL



ex2: MOV BX,[1000H]



Addressing Modes

- **Register indirect addressing (based addressing)** (can think of this as ‘base OR index only’)
  - the effective address is held in BP, BX, DI or SI.
  - eg: `MOV AX,[BX] ; MOV [BP],DL`
  - Recall: DS is used by default for BX, DI or SI;
    SS is used for BP
- Example:

```
MOV BX,1000H
MOV AX,[BX]

            AL ← DS x 10H + 1000H

            AH ← DS x 10H + 1001H
```

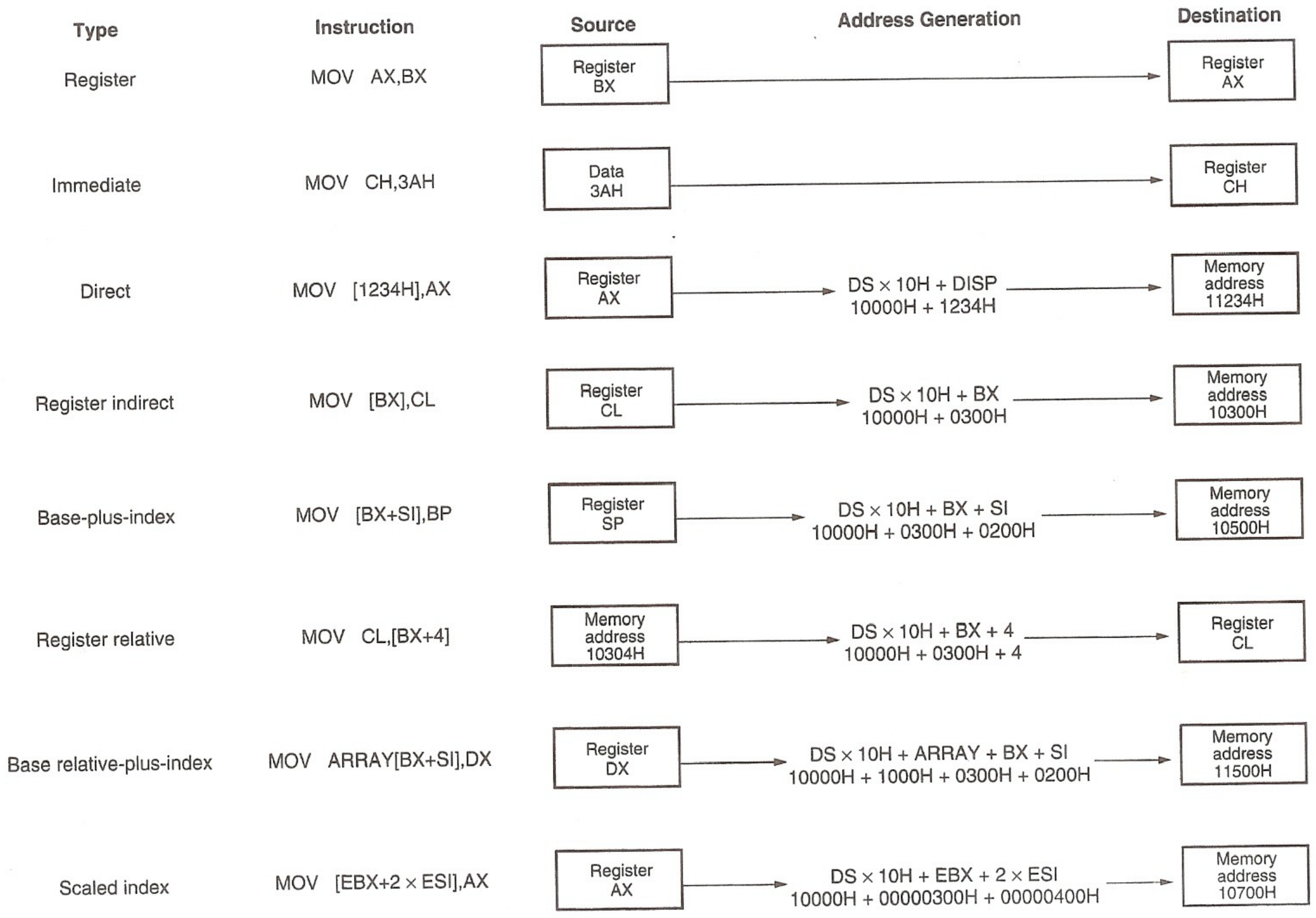
Addressing Modes

- **Register relative addressing (base + displacement)**
  - formed by the sum of a base or index register plus a displacement.
  - eg: `MOV AX,[BX+4H]`
    or: `MOV AX,4H[BX]`

- **Base plus index addressing (base + index)**
  - effective address is formed as the sum of a base register (BP or BX) and an index register (DI or SI)
  - eg: `MOV [BX+DI],CL`
  - Base register determines which segment `(e.g. [DI+BP] is relative to SS)`

Addressing Modes

- **base relative plus index addressing (base + displacement + index)**
  - effective address is the sum of base + index + displacement.
  - e.g.: `MOV [BX+DI+8AH],CL`
  - e.g.: `MOV AX,[BP+SI+ABCDH]`


Type
Instruction
Source
Address Generation
Destination
Register
MOV AX,BX
Register BX
Register AX
Immediate
MOV CH,3AH
Data 3AH
Register CH
Direct
MOV [1234H],AX
Register AX
DS × 10H + DISP
10000H + 1234H
Memory address 11234H
Register indirect
MOV [BX],CL
Register CL
DS × 10H + BX
10000H + 0300H
Memory address 10300H
Base-plus-index
MOV [BX+SI],BP
Register SP
DS × 10H + BX + SI
10000H + 0300H + 0200H
Memory address 10500H
Register relative
MOV CL,[BX+4]
Memory address 10304H
DS × 10H + BX + 4
10000H + 0300H + 4
Register CL
Base relative-plus-index
MOV ARRAY[BX+SI],DX
Register DX
DS × 10H + ARRAY + BX + SI
10000H + 1000H + 0300H + 0200H
Memory address 11500H
Scaled index
MOV [EBX+2 × ESI],AX
Register AX
DS × 10H + EBX + 2 × ESI
10000H + 00000300H + 00000400H
Memory address 10700H

Assembly and Machine Language

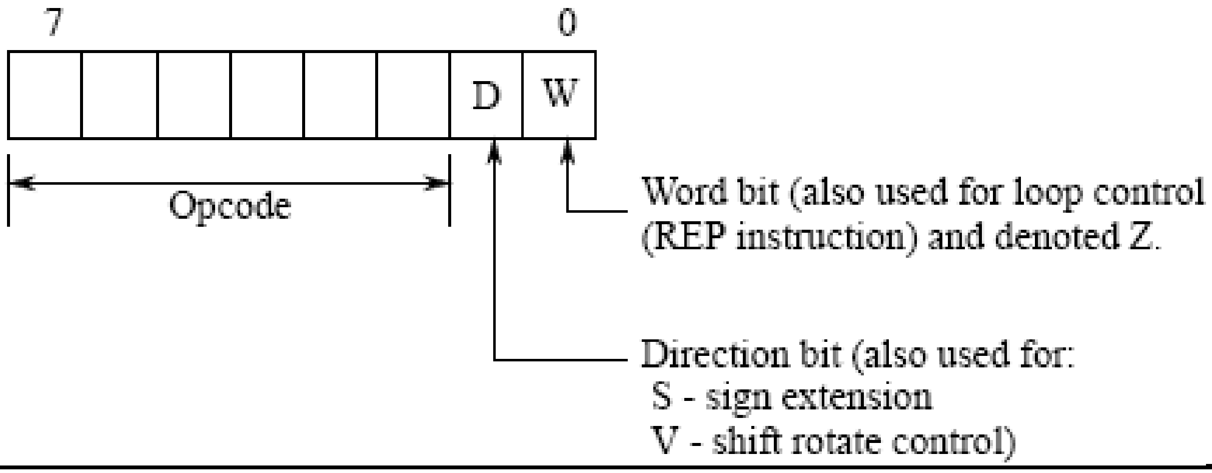
- Machine language is the native binary code that the µP understands, i.e., 1's and 0's only.
- All software, no matter what the original language was used is eventually translated to machine language for execution.
- The 8086-80286 use 16-bit mode instructions while the 80386 and up have 32-bit mode instructions (AMD has a 64 bit mode now too).
- We will focus on the 16-bit mode instructions.
  - Extensions to 32-bit mode are left as an exercise.

Assembly and Machine Language

- 16 bit mode instructions take the form:

Opcode++ 1-2 bytes	MOD-REG-R/M 0-1 byte	Displacement 0-2 bytes	Immediate 0-2 bytes

- OPCODE^{++}
  - Typically 1 byte, but not always!
  - Selects the operation (MOV, ADD, JMP)


7
0
D
W
Opcode
Word bit (also used for loop control (REP instruction) and denoted Z.
Direction bit (also used for:
S - sign extension
V - shift rotate control)

Assembly and Machine Language

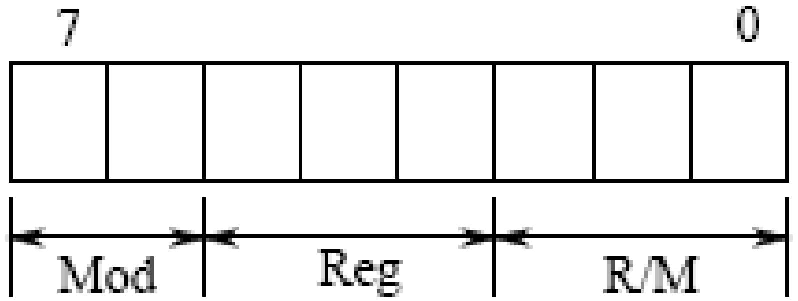
D	Direction	0	Source is specified by REG
		1	Destination is specified by REG
W	Word	0	Byte data
		1	Word data
S	Sign	0	No sign extend
		1	Sign extend 8 bit immediate to 16 bits

- Note on W & S fields:

W	S	Register	Data
0	0	8-bits	8-bits
0	1	? Sign extend to 1 byte?	
1	0	16-bits	16-bits
1	1	16-bits	8-bits

Assembly and Machine Language

- **MOD-REG-RM**



- **MOD**: Specifies addressing mode.
- **REG**: Identifies a register which is one of the instruction operands.
- **R/M**: Register/Memory coding
  - Depends on the MOD field
    - If MOD = 11, then R/M field acts as a REG field (used for register-to-register operations and other cases).
    - If MOD ≠ 11, then R/M indicates how the effective address of the operand is calculated.

Assembly and Machine Language

- MOD field:

Code	Mode	Meaning
00	Memory	No displacement (unless R/M=110)
01	Memory	8-bit displacement, will be sign extended to 16 bits
10	Memory	16-bit displacement
11	Register	R/M is a REG field

Assembly and Machine Language

- REG & R/M fields:

REG or R/M when MOD=11			R/M when MOD≠11			
REG R/M	**W=0**	**W=1**	**R/M**	**MOD=00**	**MOD=01**	**MOD=10**
000	AL	AX	000	BX+SI	BX+SI+D8	BX+SI+D16
001	CL	CX	001	BX+DI	BX+DI+D8	BX+DI+D16
010	DL	DX	010	BP+SI	BP+SI+D8	BP+SI+D16
011	BL	BX	011	BP+DI	BP+DI+D8	BP+DI+D16
100	AH	SP	100	SI	SI+D8	SI+D16
101	CH	BP	101	DI	DI+D8	DI+D16
110	DH	SI	110	*direct*	BP+D8	BP+D16
111	BH	DI	111	BX	BX+D8	BX+D16

When MOD=11,
R/M acts as a REG field

Assembly and Machine Language

- **Displacement field**
  - may be one or two bytes (language translators will generate one byte whenever possible).
  - MOD field indicates how many bytes.
  - if displacement is two bytes, the most significant byte is stored second (LITTLE endian!)
  - if displacement is one byte, the µP will ALWAYS sign-extend to 16 bits
- **Immediate field**
  - may be one or two bytes (specified by the W-bit).
  - (special case for ADD/SUB/IMUL where S-bit sign extends immediate data)
  - Little endian.

Assembly and Machine Language

- Example: Register to register addressing

```
MOV AX,BX
```

100010	D	W	MOD	REG	R/M

Opcode:	100010	
D:	1	*Dest. Specified by REG*
W:	1	*16 bit transfer*
MOD:	11	*Register in R/M*
REG:	000	*AX*
R/M:	011	*BX*

Machine instruction is:

```
1000 1011 | 1100 0011
  8    B  |   C    3
```

Assembly and Machine Language

- Example: Register to register addressing2

```
ADD AX,BX
```

000000	D	W	MOD	REG	R/M

Opcode:	000000	
D:	1	*Dest. Specified by REG*
W:	1	*16 bit transfer*
MOD:	11	*Register in R/M*
REG:	000	*AX*
R/M:	011	*BX*

Machine instruction is:

```
0000 0011 | 1100 0011
  0     3  |  C     3
```

Assembly and Machine Language

- Example: Base + index (memory) to register

```
MOV AX,[BX+DI]
```

100010	D	W	MOD	REG	R/M

Opcode:	100010	
D:	1	*Must be 1, dest AX specified by REG*
W:	1	*16 bit transfer*
MOD:	00	*No displacement*
REG:	000	*AX*
R/M:	001	*[BX+DI+DISP]*

Machine instruction is:

```
1000 1011 | 0000 0001
  8    B  |   0    1
```

Assembly and Machine Language

- Example: Base relative + index (memory) to register

```
MOV AX,[BX+DI+2H]
```

100010	D	W	MOD	REG	R/M	Displacement

Opcode:	100010	
D:	1	*Must be 1, dest AX specified by REG*
W:	1	*16 bit transfer*
MOD:	01	*8-bit displacement*
REG:	000	*AX*
R/M:	001	*[BX+DI+DISP]*

Machine instruction is:

1000	1011	0100	0001	0000	0010
8	B	4	1	0	2

Assembly and Machine Language

- Example: Base relative + index (memory) to register

```
MOV AX,[BX+DI+1234H]
```

100010	D	W	MOD	REG	R/M	Displacement

Opcode:	100010	
D:	1	*Must be 1, dest AX specified by REG*
W:	1	*16 bit transfer*
MOD:	10	*16-bit displacement*
REG:	000	*AX*
R/M:	001	*[BX+DI+DISP]*

Machine instruction is:

```
1000 1011 | 1000 0001 | 0011 0100 | 0001 0010
  8    B  |   8    1  |   3    4  |   1    2
```

Assembly and Machine Language

- Special addressing mode
  - To reference memory by displacement only (i.e. *direct* addressing mode), we use:

```
MOV [1000H],DL
MOD=00, R/M=110
```

  - From the tables (slide 19), this should correspond to [BP] with no displacement. Used for *direct* instead.
  - But what if we want to use [BP]?
    - Since [BP] cannot be used without a displacement, the assembler translates

```
MOV [BP],AL
```

to…

```
MOV [BP+0H],AL
MOD=01, R/M=110, 8-bit offset of 0H
```

Assembly and Machine Language

- Example: Immediate operand to mem/register

`MOV AX,1234H`

If W=1

1100011	W	MOD	000	R/M	data low	data high

Opcode:	1100011	MOV (imm,reg/mem)
W:	1	*16 bit transfer*
MOD:	11	*Register in R/M*
R/M:	000	*AX*
Data Low:	34H	*00110100*
Data High:	12H	*00010010*

Machine instruction is:

```
1100 0111 | 1100 0000 | 0011 0100 | 0001 0010
 C    7   |  C    0   |  3    4   |  1    2
```

Assembly and Machine Language

- Example: Immediate operand to register (not mem)

```
MOV AX,1234H
```

If W=1

1011	W	REG	data low	data high

Opcode:	1011	MOV (imm,reg)
W:	1	*16 bit transfer*
REG:	000	*AX*
Data Low:	34H	*00110100*
Data High:	12H	*00010010*

```
                       Op   WREG | DataLow   | DataHigh
Machine instruction is: 1011 1000 | 0011 0100 | 0001 0010
                        B    8    | 3    4    | 1    2
```

Note that could use general MOV imm,reg/mem but this way saves a byte

Assembly and Machine Language

- Example: Immediate operand to register2

`ADD BX,1234H`

If SW=01

100000	S	W	MOD	000	R/M	data low	data high

Opcode:	100000	ADD (imm,reg/mem)
S:	0	*Optional sign extension (can't here)*
W:	1	*16 bit transfer*
MOD:	11	*Register in R/M*
R/M:	011	*BX*
Data Low:	34H	*00110100*
Data High:	12H	*00010010*

Machine instruction is:

```
1000 0001 | 1100 0011 | 0011 0100 | 0001 0010
  8    1  |   C    3  |   3    4  |   1    2
```

Assembly and Machine Language

D	Direction	0	Source is specified by REG
		1	Destination is specified by REG
W	Word	0	Byte data
		1	Word data
S	Sign	0	No sign extend
		1	Sign extend 8 bit immediate to 16 bits

– Note on W & S fields:

W	S	Register	Data
0	0	8-bits	8-bits
0	1	? Sign extend to 1 byte?	
1	0	16-bits	16-bits
1	1	16-bits	8-bits

Assembly and Machine Language

- Example: Immediate to accumulator

```
ADD AX,1234H
```

0000010	W	data low	data high

Opcode:	0000010	ADD (imm,accum)
W:	1	*16 bit transfer*
Data Low:	34H	*00110100*
Data High:	12H	*00010010*

Machine instruction is:

```
0000 0101 | 0011 0100 | 0001 0010
 0     5  |    3   4  |  1     2
```

Note that we could have used same form as previous example, but we save a byte this way

Assembly and Machine Language

- Example: Immediate to register3

`ADD BX,-7H`

If SW=11

100000	S	W	MOD	000	R/M	data low	data high

Opcode:	`100000`	ADD (imm,mem/reg)
S:	`1`	*Sign extend to 16-bits (F9 becomes FFF9)*
W:	`1`	*16 bit transfer*
MOD:	`11`	*Register in R/M*
R/M:	`011`	*BX*
Data Low:	`F9H`	*`2's comp of 7. (will be extended to 16-bit quantity FFF9)`*

Machine instruction is:

```
1000 0011|1100 0011|1111 1001
  8    3 |  C    3 |  F    9
```

S=1: Sign extend F9 byte to FFF9 word;

S=0: Opcode becomes 8**1**C3F9**FF**

Segment Override Prefix

- Recall that `MOV AL,[BX]` uses `DS:BX` by default for calculation of the physical address
- A segment override may be given:

```
MOV AL,ES:[BX]
```

  which uses ES instead of DS for EA calc

- The machine instruction in this case includes an extra byte at the START of the instruction (i.e. lower memory):

Prefix Byte	**Segment**
26H	ES
2EH	CS
36H	SS
3EH	DS

Program Timing

- See Text Appendix B (or handout) for timing
  - Note: the times provided assume that the instructions have already been fetched and are waiting in the queue.
- Max 8086 clock:
  - 5MHz (200ns or 0.2μs per cycle)
  - 2.5MHz (400ns or 0.4μs per cycle)
- instruction times are given in clock cycles.
- Ex: Estimate the time for a 5MHz, zero wait state, 8086 to execute the following code segment:

```
        MOV DI,00FFH
AGAIN:  ADD [1234H+DI],AL
        DEC DI
        JNZ AGAIN
```

Can you calculate JNZ Displacement?

Program Timing

- Note: Loop is executed **254** times with a jump to again, and once with no jump.

Instruction	Add.Mode	T-states	Times	Total
`MOV DI,00FFH`	(reg,imm)	4	1	4
`ADD [1234H+DI],AL`	(mem,reg) EA=9	16+EA=25	255	6375
`DEC DI`	(reg 16)	3	255	765
`JNZ AGAIN`	T	16	254	4064
	F	4	1	4
TOTAL				11212

Total time is: 11212 x 200ns = 2.24ms

Note: Timing is complicated by 1) Wait States and 2) Unaligned Transfers.
These topics will be discussed later.

Reading and Problems

- Read:
  - Chapter 1 (skim protected mode)
  - Chapter 2
  - Chapter 3
  - Chapter 4, sections 1&2, skim remainder
  - Skim chapters 5-7
- Problems: see website