Intel 8086 Assembly vs Machine Language

CO 2103 Assembly Language

Topics

- Assembly vs Machine Language
- Intel 8086 Instruction Set
  - Groups
  - Instruction format
  - Addressing modes
- Translation between AL and ML
- Manual Coding

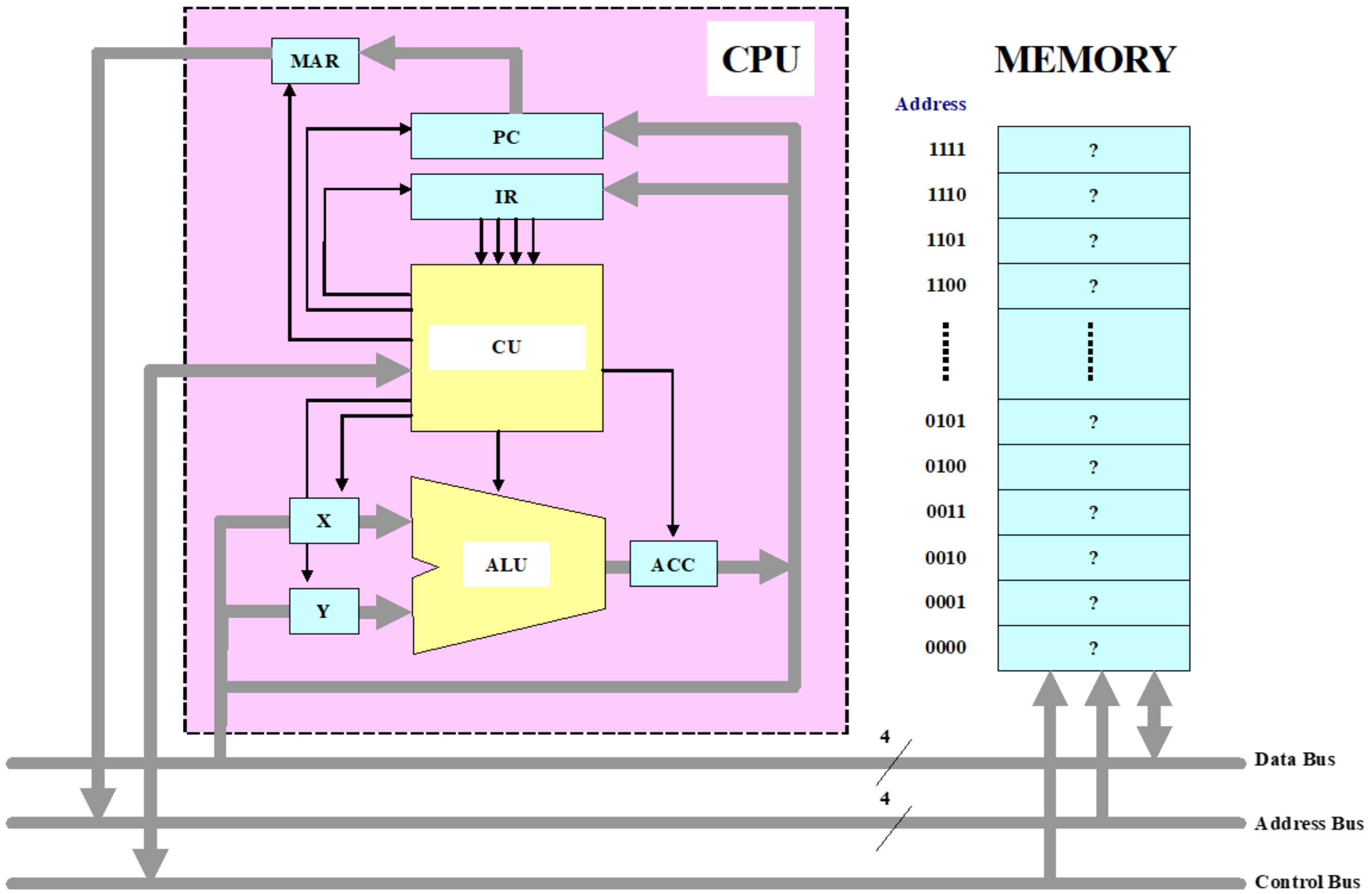
Assembly vs Machine Language

- Machine Language (Machine Codes) – what CPU understands, in 0s and 1s only; not so readable to human
  - ML is different for different CPUs
  - ML comprises of instructions in codes only
  - CPU only understand its own ML instructions
  - CPU has finite and limited ML instructions
- Assembly Language (Assembly Codes) – representation of ML in symbolic form, for improved readability to human; not readable to CPU
  - Each ML has AL one-on-one equivalence
  - AL has further high-level directives/symbols to ease AL programming job

Hypothetical CPU - 1

- Over simplified hypothetical CPU shown in next slide:
  - 4-bit Data, 4-bit Address
  - 6 registers (X, Y, ACC, IR, PC, MAR)
  - can access up to 16 memory locations (only)
  - data are stored into the memory locations through hardware (e.g. hardwired, switches) or from registers (only)
  - CU: Control Unit, ALU: Arithmetic and Logic Unit, MAR: Memory Address Register

Hypothetical CPU - 2



Hypothetical CPU – Instruction Set

- Instruction Set – list of Machine Codes or ML (hence AL) instructions that a particular CPU can understand (exclusively)
- Instruction Set analogy to Full Vocabulary of the CPU – contains what a CPU can understand only
- Assigning 0s and 1s to meaningful functions, within the CPU hardware logic

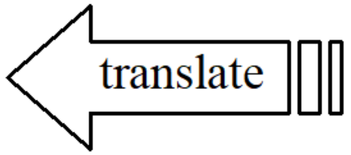
Machine Code	Instruction (AL)	Function
0000	GETX0	(X) ← (1100)
0001	GETX1	(X)← (1101)
0010	GETX2	(X) ← (1110)
0011	GETX3	(X) ← (1111)
0100	GETY0	(Y) ← (1100)
0101	GETY1	(Y) ← (1101)
0110	GETY2	(Y) ← (1110)
0111	GETY3	(Y) ← (1111)
1000	PUTA0	(1100) ← (ACC)
1001	PUTA1	(1101) ← (ACC)
1010	PUTA2	(1110) ← (ACC)
1011	PUTA3	(1111) ← (ACC)
1100	CLRX	(X) = 0
1101	CLRA	(ACC) = 0
1110	ADD	ACC ← X + Y
1111	SUB	ACC ← X – Y

Legend:

(R)	Content of Register R (R can be X, Y or ACC)
(nnnn)	Content of location nnnn

Hypothetical CPU – ML vs AL

ML: What CPU understands.	AL: What we write.
0000	GETX0
0101	GETY1
1110	ADD
1011	PUTA3



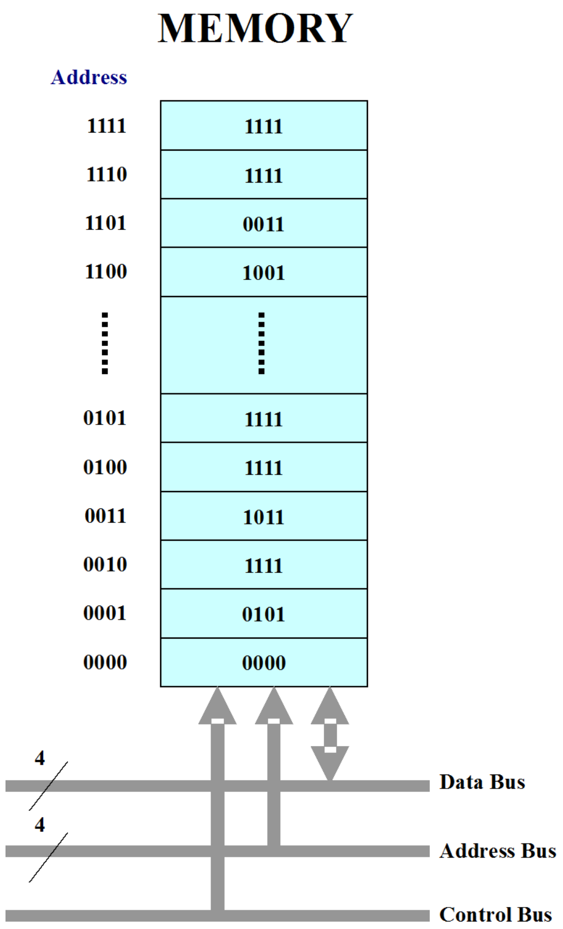
(store in memory)

Which is more readable?

- What CPU does (to execute above instructions):
  - 1 – read content in memory location 1100 to X register
  - 2 – read content in memory location 1101 to Y register
  - 3 – add register X and Y, i.e. the contents of above two memory locations, with the result in ACC register
  - 4 – save the result from ACC into memory location 1111

Hypothetical CPU – program example

- Refer to memory content shown on the right, if the CPU will start execute from location 0000 everytime it's switched ON, derive the relevant AL program and determine the new content of the memory after the CPU is switched ON? Assume the content of all other memory locations are 1111.



Hypothetical CPU – example solution

Address	Machine Code	Assembly Language	Function	Remark
0000	0000			
0001	0101			
0010	1111			
0011	1011			

above is an example of manual coding sheet

New memory content: only (1111) changed from 1111 to 0100

Instruction Set

- "An instruction set is a list of all the instructions, and all their variations, that a processor (or in the case of a virtual machine, an interpreter) can execute." – Wikipedia
  - A particular processor can ONLY understand the instructions available in its instruction set
  - Analogy to the complete vocabulary of a particular language; the dictionary

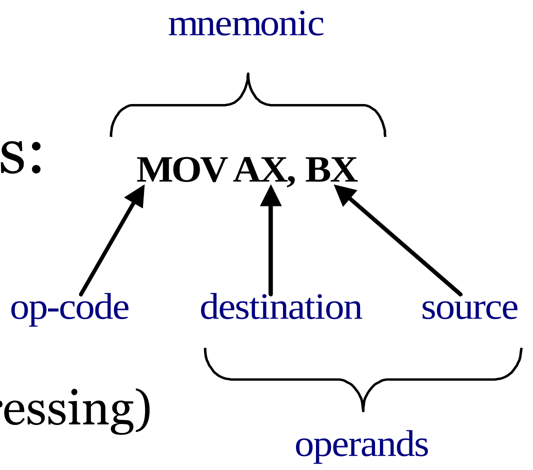
Intel 8086 Instruction Set

- 137 instructions only (on separate sheets)
- The instructions can be grouped based on their functions:
  - data transfer – MOV, LEA, LDS, LES, XCHG, XLAT
  - input-output – IN, OUT
  - logical – NOT, AND, OR, XOR, TEST
  - shift and rotate – SHL, SHR, SAL, SAR, ROL, ROH, RCL, RCH
  - arithmetic – ADD, SUB, ADC, SBB, INC, DEC, NEG, CMP, MUL, DIV, IMUL, IDIV, CBW, CWD, AAA, AAS, AAM, AAD, DAA, DAS
  - program control – JMP, all other Jumps, LOOP, LOOPE, LOOPNE, LOOPZ, LOOPNZ, JCXZ
  - subroutine and interrupt – CALL, RET, INT, INTO, IRET
  - string – MOVS, MOVSB, MOVSW, CMPS, SCAS, LODS, STOS, REP, REPE, REPZ, REPNE, REPNZ
  - processor control – STC, CLC, CMC, STD, CLD, STI, CLI, LAHF, SAHF, ESC, LOCK, NOP, WAIT, HALT
  - stack – PUSH, POP, PUSHF, POPF
- Grouping of instructions is not standardized

Format of AL Instructions

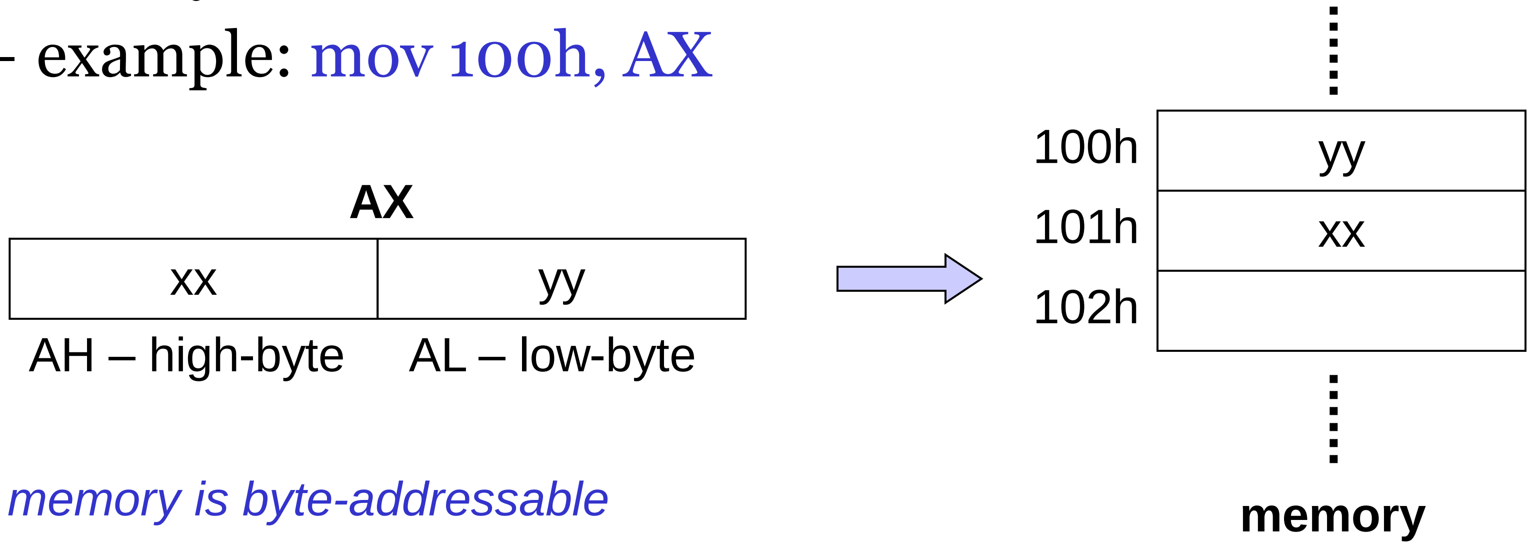
- Format:
  - each AL instruction (symbolic representation of Machine Code) is called Mnemonic
  - each Mnemonic comprises of two parts: op-code and operands (optional); many articles define mnemonic as the op-code
  - if there can be up to 2 operands, the first is the destination, while the last is the source operand

- Each AL instruction may be 1 to 6 bytes:
  - op-code – 1 or 2 bytes
  - operand(s) can be:
    - 2-byte offset (for direct addressing)
    - 1- or 2-byte displacement (for indexed addressing)
    - 1- or 2-byte immediate operand
    - 4-byte physical address (for intersegment jump or procedure call)



Reminder on Memory Access

- For Intel 8086, memory accesses are in Little-Endian system. The low-byte will be transferred to/from the lower-address memory location, while high-byte to/from the higher-address memory location.
  - example: mov 100h, AX



Note: memory is byte-addressable

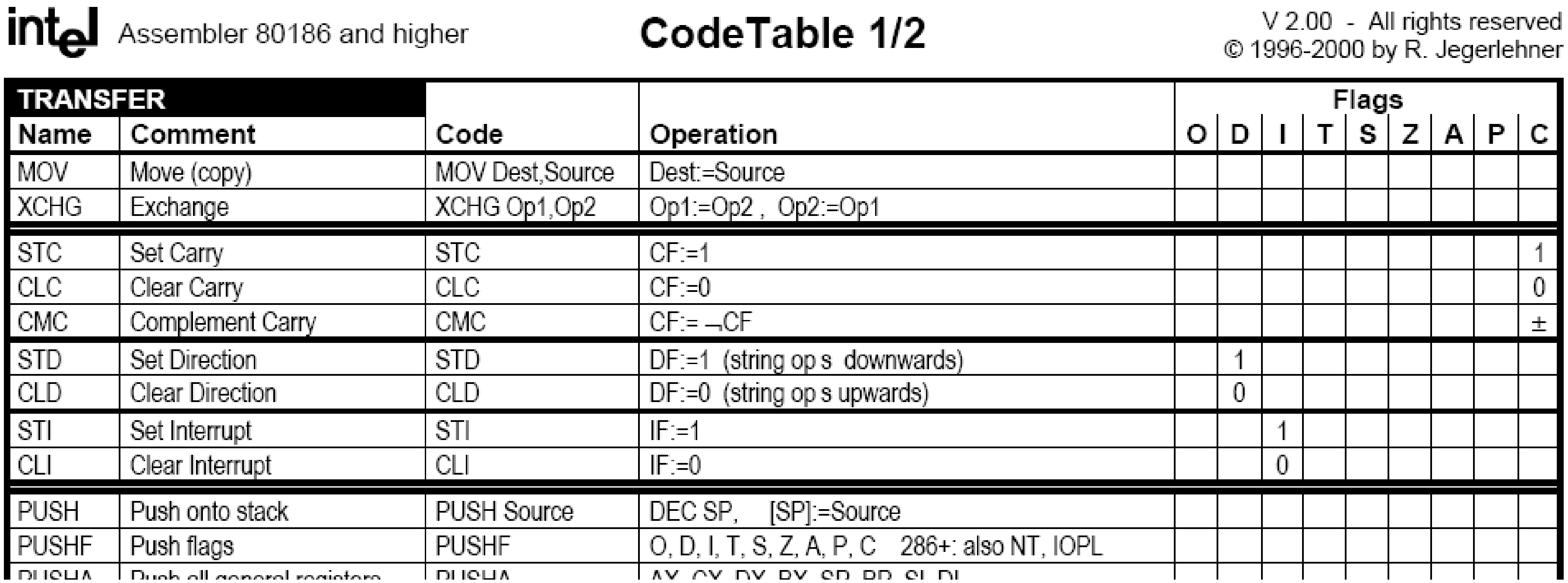
Looking up Instruction - 1

- Information to check:
  - function
  - possible operand(s)
  - effect on flags
  - others: machine code, number of byte, clock cycles
- Example:
  - REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP or SP
  - SREG: DS, ES, SS, or only as second operand: CS
  - memory: [BX], [BX+SI+7], variable, etc

Opcode	Operands	Description
MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate SREG, memory memory, SREG REG, SREG SREG, REG	Function: Operand1 $\leftarrow$ Operand2 Function in words: Copy Operand2 to Operand1 Algorithm: Operand1 = Operand2 Flags: C Z S O P A unchanged

Looking up Instruction - 2

- Alternative Instruction Set format …



intel Assembler 80186 and higher **CodeTable 1/2** V 2.00 - All rights reserved © 1996-2000 by R. Jegerlehner

TRANSFER				**Flags**								
Name	**Comment**	**Code**	**Operation**	**O**	**D**	**I**	**T**	**S**	**Z**	**A**	**P**	**C**
MOV	Move (copy)	MOV Dest,Source	Dest:=Source									
XCHG	Exchange	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1									
STC	Set Carry	STC	CF:=1									1
CLC	Clear Carry	CLC	CF:=0									0
CMC	Complement Carry	CMC	CF:= ¬CF									±
STD	Set Direction	STD	DF:=1 (string op s downwards)		1							
CLD	Clear Direction	CLD	DF:=0 (string op s upwards)		0							
STI	Set Interrupt	STI	IF:=1			1						
CLI	Clear Interrupt	CLI	IF:=0			0						
PUSH	Push onto stack	PUSH Source	DEC SP, [SP]:=Source									
PUSHF	Push flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL									
PUSHA	Push all general registers	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI									

Addressing Modes - 1

- Addressing Modes – methods for specifying the operand(s) for a Machine Code instruction
  - the way the location of the data is being specified
  - the way to address the operand(s)
- Five (5) modes for 8086:
  - immediate
  - register
  - direct
  - indirect
  - indexed (base, index, base-index)

Addressing Modes - 2

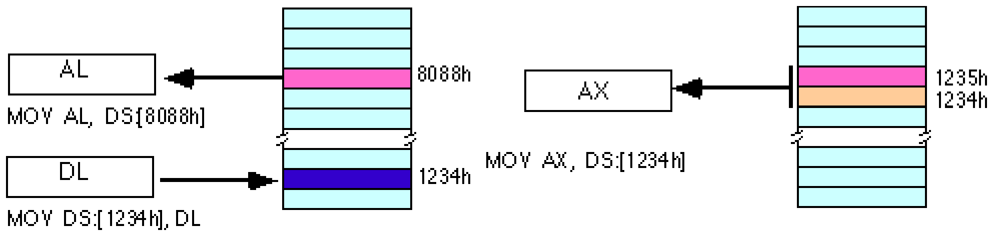
- Immediate – the data is encoded as part of the instruction, e.g.
  - ADD AX, 2 ; add 2 to AX
- Register – the operand is inside a register, e.g.
  - ADD AX, BX ; add AX to BX
- Direct – the memory address of the data is given as part of the instruction, e.g.
  - ADD AX, [2] ; add content of location 2 to AX
- Indirect – the effective address (offset) of the data is stored using a base or index register (BX, BP, SI, DI), e.g.
  - ADD AX, [BX] ; add content of location whose address is stored in BX, to AX
  - MOV [BP], DL, ADD AX, [DI], etc

Addressing Modes - 3

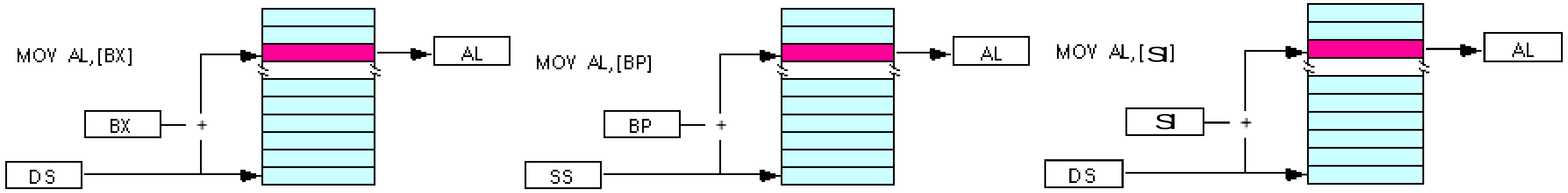
- Relative Indirect – the effective address (offset) is calculated:
  - indexed – indirect addressing and a constant displacement, e.g. ADD AX, [SI+2]
  - base-indexed – base and index registers, e.g. ADD AX, [BP+SI]
  - based-indexed +displacement – base and index register and a constant displacement, e.g. ADD AX, [BX+DI+2]
- An instruction can use more than one AM, e.g.
  - ADD AX, 2 uses register and immediate addressing modes; however, we usually call this immediate

AM – examples 1

- Direct AM
  - uses data segment (DS) by default, however, can use segment override

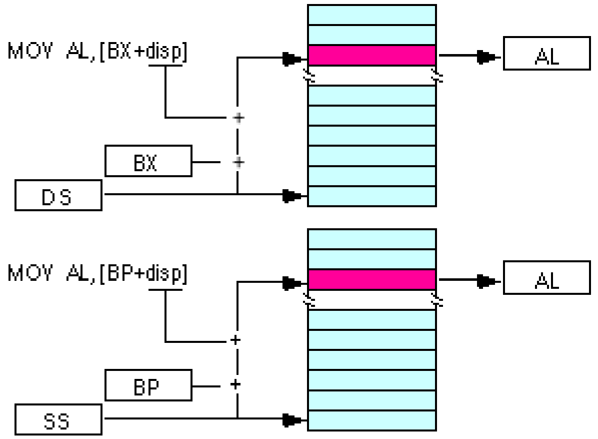


- Indirect AM
  - Base AM: [BX] uses DS by default, [BP] uses SS by default
  - Indexed AM: [SI] and [DI] use DS by default
  - can use segment override, e.g. MOV AL, CS:[BX]

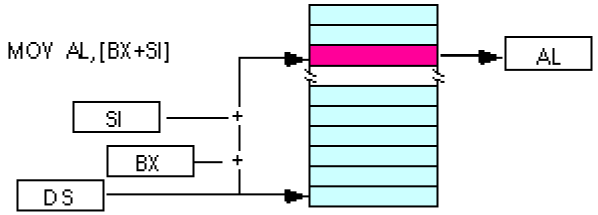


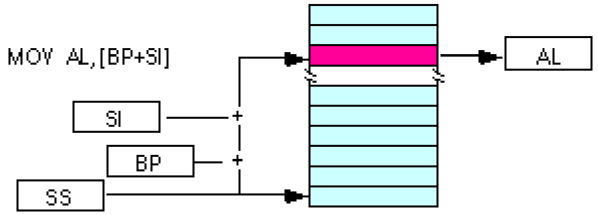
AM – examples 2

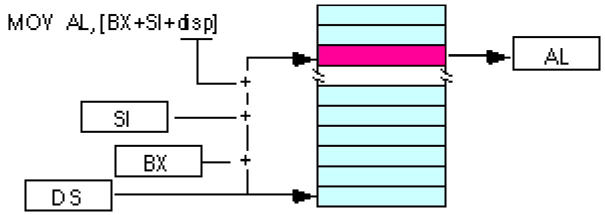
- Relative Indirect AM
  - similarly for [SI] and [DI]

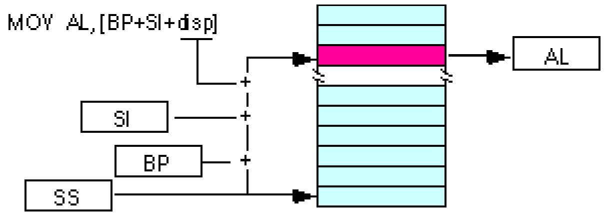


- Base-indexed AM
  - similarly for [DI]









Determining the EA, PA - 1

- Effective Address – offset address into the target segment
- System for examples in next slide (numbers are in Hex):

Data registers

	8-bit (Hi)	8-bit (Lo)
AX	17	FE
BX	3C	AB
CX	00	E0
DX	FF	FF

Pointer/index registers

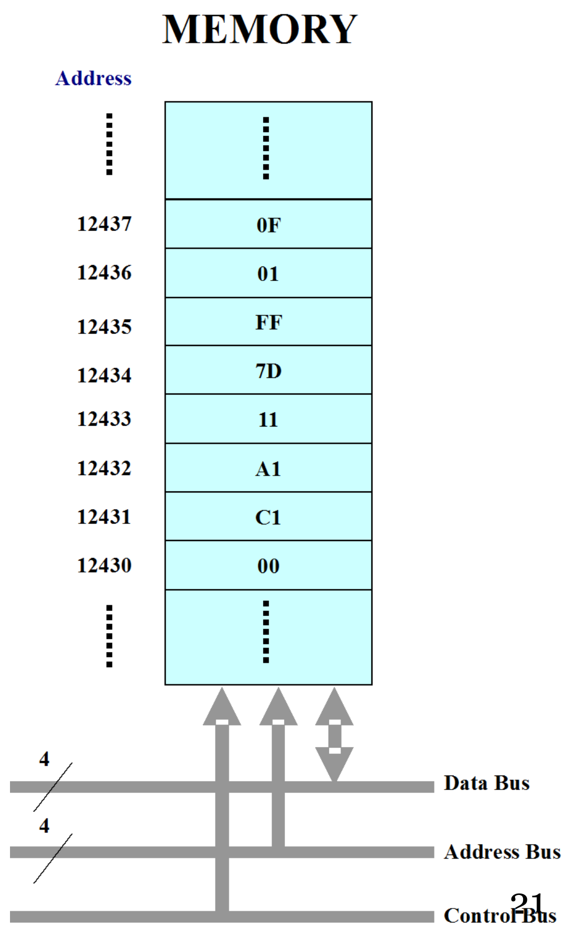
SP	00E0
BP	1220
SI	1234
DI	0010
IP	0C39

Segment registers (16-bit)

CS	F000
SS	C000
DS	1120
ES	A000

Status register

PSW



Determining the EA, PA - 2

- Examples (refer to diagram in previous slide)
  - MOV AX, [SI]
    - EA = DS:1234, PA = 11200+1234 = 12434
    - (AL) ← (12434), (AH) ← (12435), i.e. (AX) = FF 7D
  - MOV [DI+2], BL
    - EA = DS:[0010+2] = DS:0012,

      PA = 11200+0012 = 11212
    - (11212) ← (BL), i.e. (11212) = AB
  - MOV AL, [BP][DI]3
    - EA = DS:[1220+0010+3] = DS:1233,

      PA = 11200+1233 = 12433
    - (AL) ← (12433), i.e. (AL) = 11

Syntax for AM

- Assembler dependent
  - disp[bx], [bx][disp], [bx+disp], [disp][bx], and [disp+bx]
  - [bx][si], [bx+si], [si][bx], and [si+bx]
  - disp[bx][si], disp[bx+si], [disp+bx+si], [disp+bx][si], disp[si][bx], [disp+si][bx], [disp+si+bx], [si+disp+bx], [bx+disp+si], etc.
- 17 legal memory AM: disp, [bx], [bp], [si], [di], disp[bx], disp[bp], disp[si], disp[di], [bx][si], [bx][di], [bp][si], [bp][di], disp[bx][si], disp [bx][di], disp[bp][si], and disp[bp][di]
  - to remember: pick one or none from each column

	[BX]	[SI]
DISP		
	[BP]	[DI]
displacement	base	index

AM Summary

Addressing Mode	Operand	Default Segment
Immediate	Data	None
Register	Register	None
Direct	[Offset]	DS
Indirect	[BX], [SI], [DI]	DS
Based Relative	[BX]+disp, [BP]+disp	DS SS
Indexed Relative	[DI]+disp, [SI]+disp	DS
Based Indexed	[BX][DI or SI]+disp, [BP][DI or SI]+disp	DS SS

Registers Assignment for Addressing

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	None	IP
Stack Operation	SS	None	SP, BP
General Data	DS	CS, ES, SS	BX, address
String Source	DS	CS, ES, SS	SI, DI, address
String Destination	ES	None	DI

ML / AL Translation - 1

- An instruction can be encoded into 1 to 6 bytes
- First two bytes are for the op-code, while remaining bytes are for the operands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						***d***	***s***	***mod***		***reg***			***reg/mem***		

- Byte 1 contains 3 kinds of information:
  - Opcode field (6-bit) specifies the instruction class, i.e. MOV, ADD, SUB, etc
  - Register Direction bit (D bit) tells whether the register operand in REG field is source or destination operand
    - D=0: data flows from REG to R/M
    - D=1: data flows to REG from R/M

ML / AL Translation - 2

  - Data Size bit (S bit) specifies whether the operation will be performed on 8-bit or 16-bit data
    - S=0: 8-bit
    - S=1: 16-bit
  - Bit 8 and 9 will take different meaning when dealing with immediate addressing mode (see Example 4)
- Byte 2 has 3 fields:
  - Mode field (MOD) – 2 bits
  - Register field (REG) – 3 bits
  - Register/Memory field (R/M) – 3 bits
- REG field specifies the first operand

REG	s=0	s=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

ML / AL Translation - 3

- 2-bit MOD field and 3-bit R/M field together specify the second operand

MOD	Explanation
00	Memory mode, no displacement follows Except when R/M=110, then 16-bit displacement follows
01	Memory mode, 8-bit displacement follows
10	Memory mode, 16-bit displacement follows
11	Register mode, no displacement

MOD=11			Effective Address Calculation			
R/M	S=0	S=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 Address	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

ML / AL Translation – example 1

SUB AX, AX

2B C0 = 0010 1011 1100 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						***d***	***S***	***mod***		***reg***			***reg/mem***		
0	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0
2				B				C				0			

```
opcode = 001010 : Subtract reg/mem from reg
     d =  1 : to REG
     s =  1 : word (16 bits)
   mod = 11 : reg/mem is register field
```

ML / AL Translation – example 2

ADD AX, [BX]

03 07 = 0000 0011 0000 0111

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						***d***	***s***	***mod***		***reg***			***reg/mem***		
0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1
0				3				0				7			

```
opcode = 000000 : Add reg/mem to reg
     d =  1 : to REG
     s =  1 : word (16 bits)
   mod = 00 : displacement = 0, unless reg/mem = 110
```

ML / AL Translation – example 3

```
CMP DX,BX

3B D3 = 0011 1011 1101 0011
```

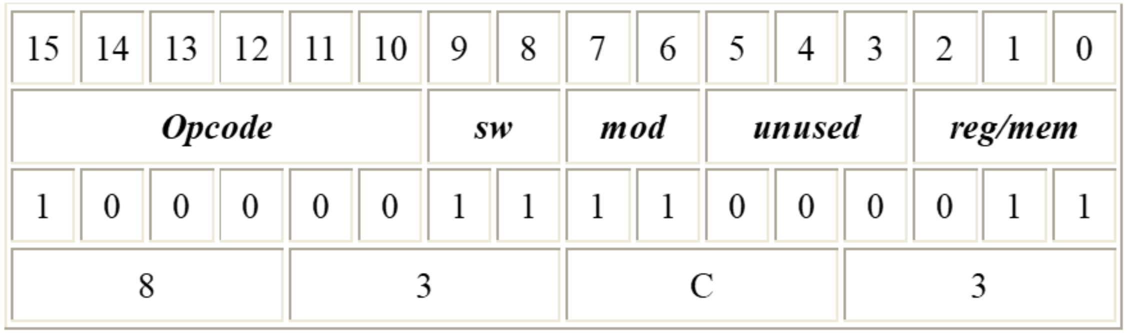
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						***d***	***s***	***mod***		***reg***			***reg/mem***		
0	0	1	1	1	0	1	1	1	1	0	1	0	0	1	1
3				B				D				3			

```
 opcode = 001110 : Compare reg/mem with reg
      d =  1 : to REG
      s =  1 : word (16 bits)
    mod = 11 : reg/mem is register field
    reg = destination
reg/mem = source
```

ML / AL Translation – example 4

ADD BX, 2

83 C3 02 = 1000 0011 1100 0011 0000 0010



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode						***sw***		***mod***		***unused***			***reg/mem***		
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1
8				3				C				3			

```
 opcode = 100000 : Add immediate (MSB=1) to reg/mem
     sw = 01 : 16 bits of immediate data form the operand
          11 :  8 bits of immediate data is sign-extended
                         to form the 16-bit operand
    mod = 11 : reg/mem is register field
 unused = 000
reg/mem = destination

The 3rd or 3rd & 4th byte is the immediate source operand
```

Manual Coding - 1

- Manual Coding is the process of writing AL program, and manually encode the AL into ML (Machine Codes) and assigning memory locations for the storage of the codes
  - uses Manual Coding sheet, which usually comprises of Memory Address (where the code is stored – may be more than 1 byte for an instruction), Machine Codes, AL Instructions and Functions

Address (Hex)	Machine Code (Hex)	Instructions	Function	Remark
		¦		
F02C	C3	MOV BL, AL	(BL) ← (AL)	(BL) = nn
F02D	88			

  - function can be described in symbolic, e.g. (BL) ← (AL) means content of AL copied into BL

Manual Coding - 2

- Manual coding is tedious exercise
- There are tools to perform the translation of AL to ML and then assigning the ML into appropriate memory locations
- We will look into these tools in next Chapter …

Sample 8086 AL Program

```
; This program displays "Hello, world!"
.model small
.data
message db "Hello, world!",0dh,0ah,'$'        ;newline + eoc
.code
main:   mov ax,@data                ; data segment
mov ds,ax
mov ah,9
mov dx,offset message  ; display msg starting at 0
int 21h
mov ax,4c00h           ; halt the program and return
int 21h
end main
```

- Note some high-level structure/notations

Equivalent ML Program

- ML Program of AL program in previous slide:

```
0B40:0000  48 65 6C 6C 6F 2C
           20 77 6F 72 6C 64
           21 0D 0A 24

0B50:0000  B8 40 0B
0B50:0003  8E D8
0B50:0005  B4 09
0B50:0007  BA 00 00
0B50:000A  CD 21
0B50:000C  B8 00 4C
0B50:000F  CD 21
```

- It is not easy to read/understand
- The above in Hex. Imagine them in binary – unreadable.

Summary

- ML instructions are in zeros and ones that can be directly feed to logic circuits to perform the operations
- AL improves readability of ML: symbolic representation
- Essence of AL programming:
  - know the CPU – registers
  - know the memory
  - know the Instruction Set
- Format of AL instruction:
  - Mnemonic = Opcode + Operands
- Intel 8086 Instruction Set: 137 instructions, 10 groups
- Addressing Modes: Immediate, Register, Direct, Indirect, Relative Indirect
- Encoding AL to ML: making sense out of the 0s and 1s