



# معماری و سازمان کامپیوتر

دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

امیر خورسندی

بهار ۱۴۰۲

# برنامه نویسی کامپیوتر پایه

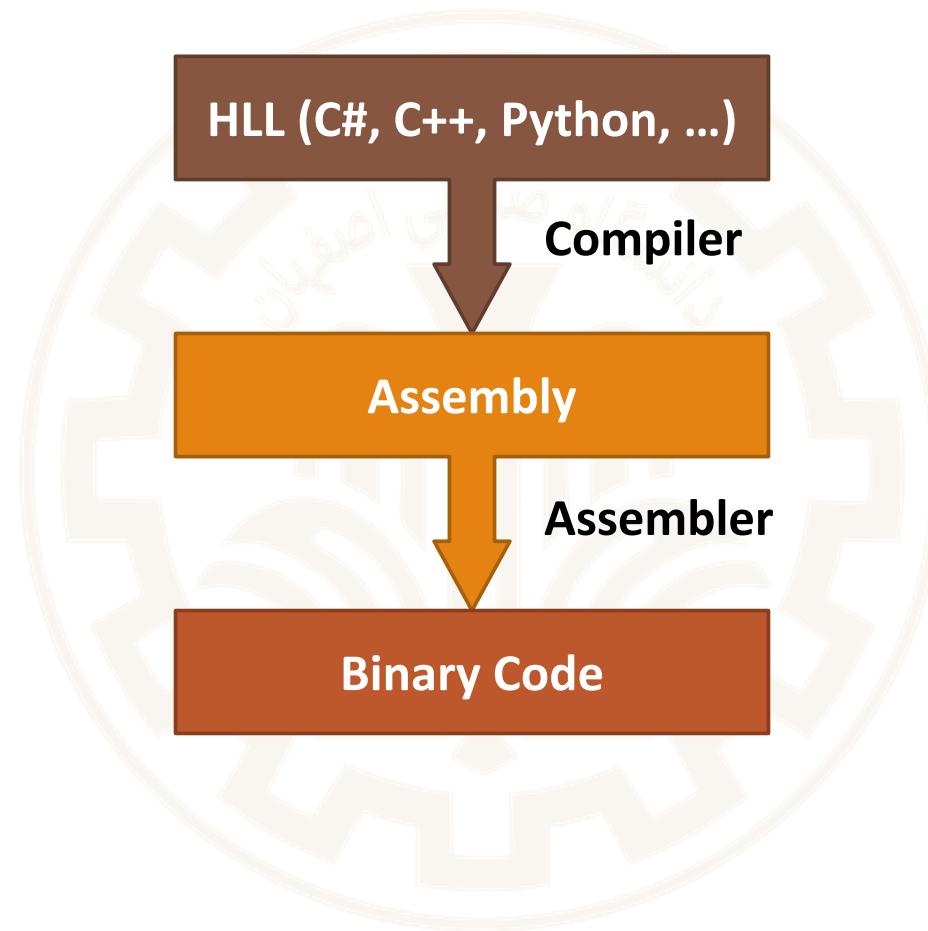
# برنامه کامپیوتری

- تعدادی دستورالعمل متوالی که برای انجام هدفی خاص اجرا می گردند.
- بررسی ساختار برنامه جزء معماری کامپیوتر نمی باشد.
- اما داشتن اطلاعات از معماری کامپیوتر به برنامه نویس کمک می کند.

# انواع برنامه

- برنامه زبان ماشین: کد باینری
- برنامه اسمبلی: مجموعه سمبل ها، علائم و رقم ها
- برنامه زبان سطح بالا: نزدیک به زبان معمول و مستقل از سخت افزار

# روند تبدیل برنامه



# زبان اسمبلی کامپیوتر پایه

• هر دستور از سه بخش تشکیل می شود:

۱. برچسب دستور (،)
۲. عملیاتی که قرار است انجام شود.
۳. توضیحات دستور

Label	Instruction	Comment
-------	-------------	---------

# دستورات اصلی زبان اسمبلی

• همان دستورات سمبلیک کامپیوتر مبنا

۱. مراجعه به حافظه

• دستور

• آدرس عملوند

• حرف I

۲. مراجعه به ثبات

۳. مراجعه به ورودی/خروجی

# شبه دستورات زبان اسمبلی

- کار پردازشی انجام نمی دهند و فقط به عنوان راهنما استفاده می شوند:
- **ORG N**: محلی از حافظه که ادامه برنامه در آن قرار دارد.
- **END**: پایان برنامه
- **DEC N**: عملوند به صورت عدد اعشاری
- **HEX N**: عملوند به صورت عدد هگزادسیمال



# مثال

• برنامه تفریق دو عدد در حافظه

	ORG 100	/Origin of program is location 100
	LDA SUB	/Load subtrahend to AC
	CMA	/Complement AC
	INC	/Increment AC
	ADD MIN	/Add minuend to AC
	STA DIF	/Store difference
	HLT	/Halt computer
MIN,	DEC 83	/Minuend
SUB,	DEC -23	/Subtrahend
DIF,	HEX 0	/Difference stored here
	END	/End of symbolic program

# ترجمه به زبان ماشین

Hexadecimal code		
Location	Content	Symbolic program
		ORG 100
100	2107	LDA SUB
101	7200	CMA
102	7020	INC
103	1106	ADD MIN
104	3108	STA DIF
105	7001	HLT
106	0053	MIN, DEC 83
107	FFE9	SUB, DEC -23
108	0000	DIF, HEX 0
		END

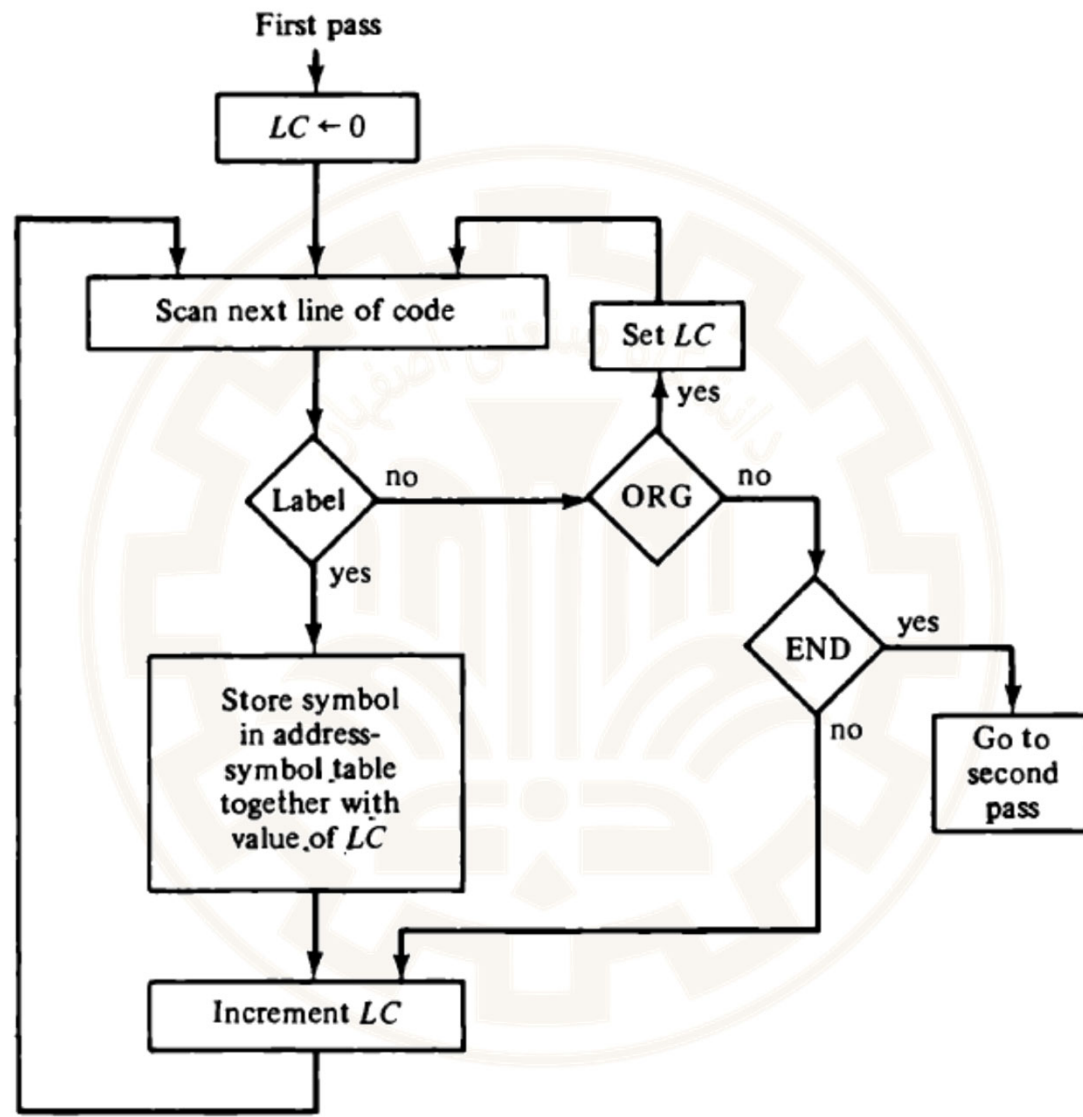
# مراحل ترجمه

• در دو مرحله برنامه پیمایش می شود:

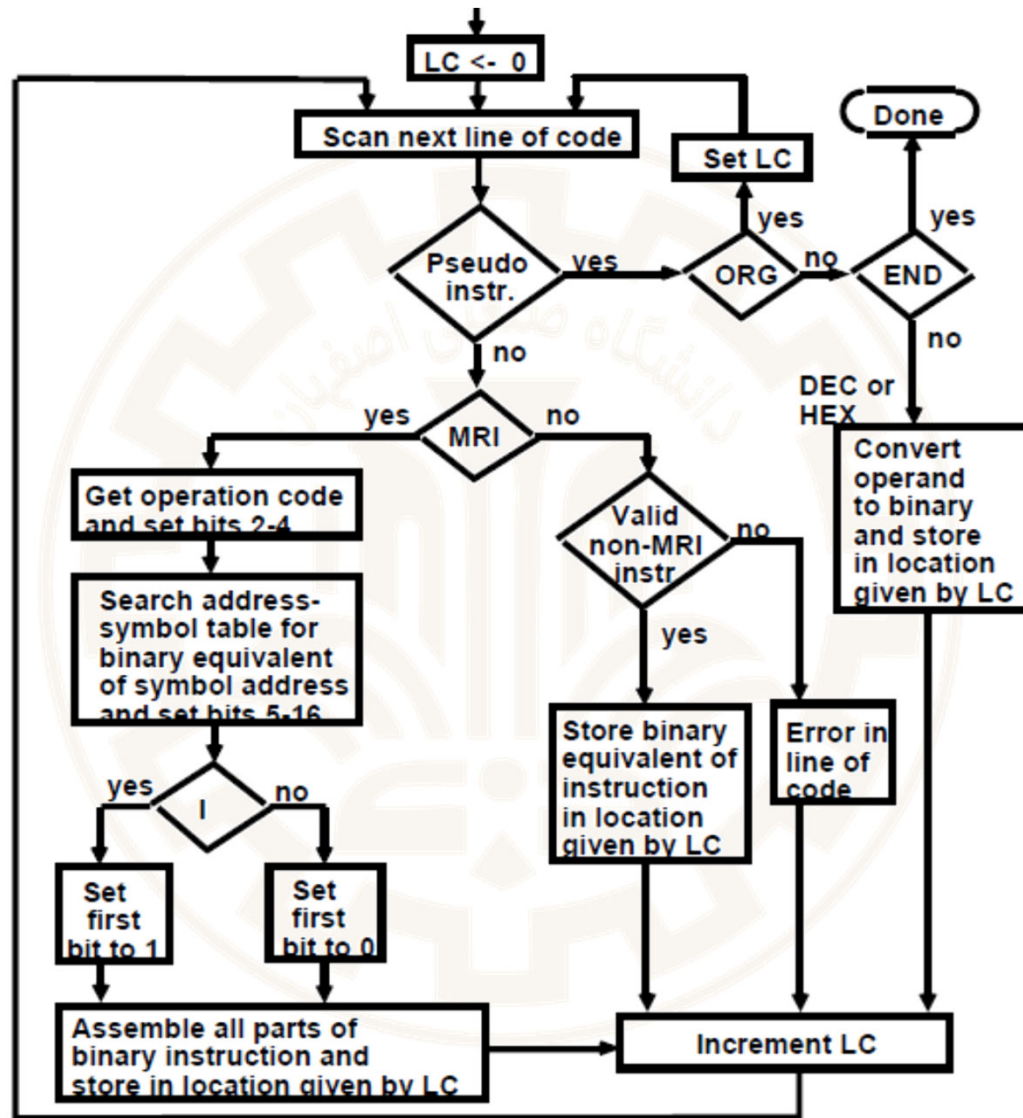
۱. در مرحله اول جدول سمبل ها به همراه آدرس ها به دست می آید.

۲. در مرحله دوم با استفاده از جدول سمبل ها ترجمه انجام می شود.

# مرحله نخست ترجمه



## مرحله دوم ترجمه



# پیاده سازی حلقه

• جمع ۱۰۰ عدد

	ORG 100	/Origin of program is HEX 100
	LDA ADS	/Load first address of operands
	STA PTR	/Store in pointer
	LDA NBR	/Load minus 100
	STA CTR	/Store in counter
	CLA	/Clear accumulator
LOP,	ADD PTR I	/Add an operand to AC
	ISZ PTR	/Increment pointer
	ISZ CTR	/Increment counter
	BUN LOP	/Repeat loop again
	STA SUM	/Store sum
	HLT	/Halt
ADS,	HEX 150	/First address of operands
PTR,	HEX 0	/This location reserved for a pointer
NBR,	DEC -100	/Constant to initialize counter
CTR,	HEX 0	/This location reserved for a counter
SUM,	HEX 0	/Sum is stored here
	ORG 150	/Origin of operands is HEX 150
	DEC 75	/First operand
	DEC 23	/Last operand
	END	/End of symbolic program

# عملیات منطقی

• OR کردن دو مقدار

LDA A      Load first operand  $A$   
CMA        Complement to get  $\bar{A}$   
STA TMP    Store in a temporary location  
LDA B      Load second operand  $B$   
CMA        Complement to get  $\bar{B}$   
AND TMP    AND with  $\bar{A}$  to get  $\bar{A} \wedge \bar{B}$   
CMA        Complement again to get  $A \vee B$



# عملیات شیف

• شیفت به راست

CLE  
CIR

• شیفت به چپ

CLE  
CIL

• شیفت ریاضی به راست

CLE /Clear  $E$  to 0  
SPA /Skip if AC is positive;  $E$  remains 0  
CME /AC is negative; set  $E$  to 1  
CIR /Circulate  $E$  and AC



# انجام محاسبات ریاضی

- اگر در سخت افزار دستور مجزا برای پردازش مورد نظر باشد استفاده خواهد شد.
- در غیر این صورت باید برنامه ای نوشته شود که بر اساس دستورات موجود پردازش را انجام دهد.

# جمع اعداد بزرگ

• جمع دو عدد ۳۲ بیتی

LDA AL	/Load A low
ADD BL	/Add B low, carry in E
STA CL	/Store in C low
CLA	/Clear AC
CIL	/Circulate to bring carry into AC(16)
ADD AH	/Add A high and carry
ADD BH	/Add B high
STA CH	/Store in C high
HLT	

AL,	—	/Location of operands
AH,	—	
BL,	—	
BH,	—	
CL,	—	
CH,	—	

# ضرب دو عدد

	ORG 100	
LOP,	CLE	/Clear E
	LDA Y	/Load multiplier
	CIR	/Transfer multiplier bit to E
	STA Y	/Store shifted multiplier
	SZE	/Check if bit is zero
	BUN ONE	/Bit is one; go to ONE
	BUN ZRO	/Bit is zero; go to ZRO
ONE,	LDA X	/Load multiplicand
	ADD P	/Add to partial product
	STA P	/Store partial product
	CLE	/Clear E
ZRO,	LDA X	/Load multiplicand
	CIL	/Shift left
	STA X	/Store shifted multiplicand
	ISZ CTR	/Increment counter
	BUN LOP	/Counter not zero; repeat loop
	HLT	/Counter is zero; halt
CTR,	DEC -8	/This location serves as a counter
X,	HEX 000F	/Multiplicand stored here
Y,	HEX 000B	/Multiplier stored here
P,	HEX 0	/Product formed here
	END	

- حاصل ضرب دو عدد ۸ بیتی  
شانزده بیت خواهد بود.

- یک قطعه مجزا از برنامه که به صورت مکرر استفاده خواهد شد.
- ذخیره آدرس برگشت قبل از شروع اجرای تابع
- برگشت به ادامه اجرای برنامه اصلی در انتهای تابع

# تابع شیفت

• شیفت ۴ بیتی اکومولاتور

	ORG 100	/Main program
	LDA X	/Load X
	BSA SH4	/Branch to subroutine
	STA X	/Store shifted number
	LDA Y	/Load Y
	BSA SH4	/Branch to subroutine again
	STA Y	/Store shifted number
	HLT	
X,	HEX 1234	
Y,	HEX 4321	
		/Subroutine to shift left 4 times
SH4,	HEX 0	/Store return address here
	CIL	/Circulate left once
	CIL	
	CIL	
	CIL	/Circulate left fourth time
	AND MSK	/Set AC(13–16) to zero
	BUN SH4 I	/Return to main program
MSK,	HEX FFF0	/Mask operand
	END	

# ارسال پارامتر ورودی به توابع

- یک پارامتر را می توان از طریق اکومولاتور انتقال داد.
- سایر پارامترها را باید از طریق حافظه منتقل کرد.
- اولین پارامتر بلافاصله پس از فراخوانی تابع در برنامه اصلی تعیین می شود.

# تابع OR

ORG 200  
LDA X /Load first operand into AC  
BSA OR /Branch to subroutine OR  
HEX 3AF6 /Second operand stored here  
STA Y /Subroutine returns here  
HLT  
X, HEX 7B95 /First operand stored here  
Y, HEX 0 /Result stored here  
OR, HEX 0 /Subroutine OR  
CMA /Complement first operand  
STA TMP /Store in temporary location  
LDA OR I /Load second operand  
CMA /Complement second operand  
AND TMP /AND complemented first operand  
CMA /Complement again to get OR  
ISZ OR /Increment return address  
BUN OR I /Return to main program  
TMP, HEX 0 /Temporary storage  
END



# تابع انتقال اطلاعات

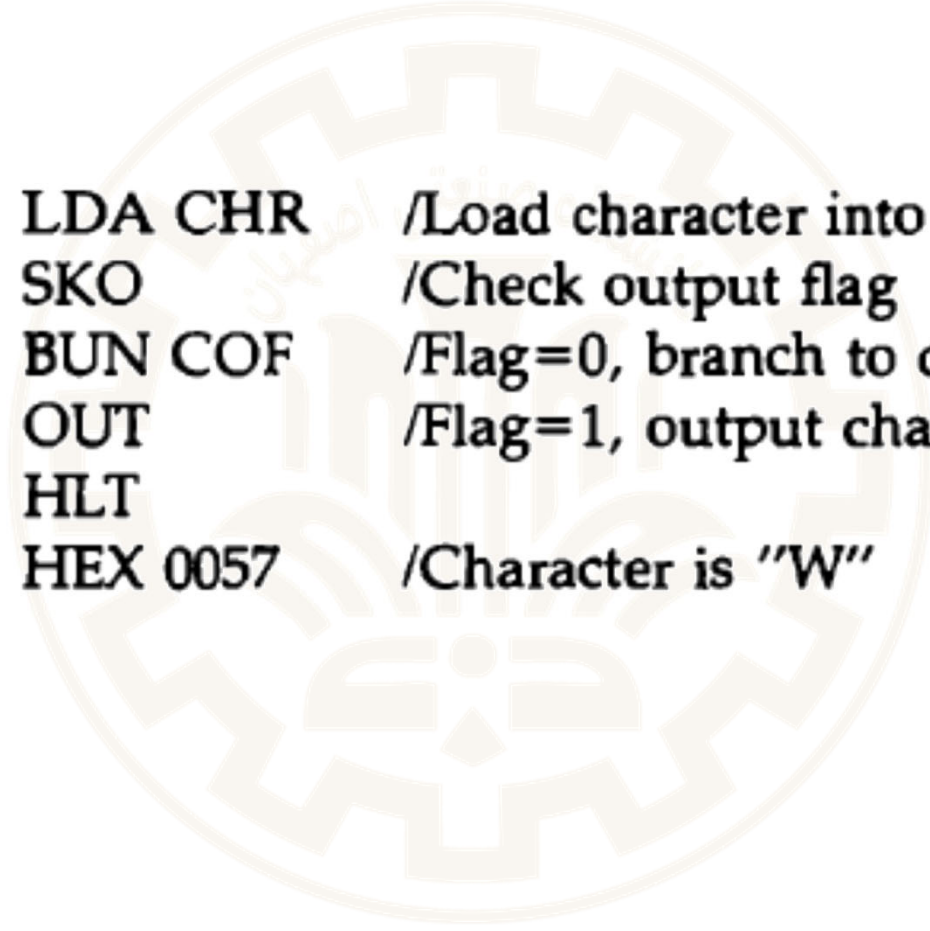
	BSA MVE	/Main program
	HEX 100	/Branch to subroutine
	HEX 200	/First address of source data
	DEC -16	/First address of destination data
	HLT	/Number of items to move
MVE,	HEX 0	/Subroutine MVE
	LDA MVE I	/Bring address of source
	STA PT1	/Store in first pointer
	ISZ MVE	/Increment return address
	LDA MVE I	/Bring address of destination
	STA PT2	/Store in second pointer
	ISZ MVE	/Increment return address
	LDA MVE I	/Bring number of items
	STA CTR	/Store in counter
	ISZ MVE	/Increment return address
LOP,	LDA PT1 I	/Load source item
	STA PT2 I	/Store in destination
	ISZ PT1	/Increment source pointer
	ISZ PT2	/Increment destination pointer
	ISZ CTR	/Increment counter
	BUN LOP	/Repeat 16 times
	BUN MVE I	/Return to main program
PT1,	—	
PT2,	—	
CTR,	—	



# خواندن از ورودی

CIF,	SKI	/Check input flag
	BUN CIF	/Flag=0, branch to check again
	INP	/Flag=1, input character
	OUT	/Print character
	STA CHR	/Store character
	HLT	
CHR,	—	/Store character here

# ارسال به خروجی



.	LDA CHR	/Load character into AC
COF,	SKO	/Check output flag
	BUN COF	/Flag=0, branch to check again
	OUT	/Flag=1, output character
	HLT	
CHR,	HEX 0057	/Character is "W"

# تابع وقفه

۱. محتوای ثبات ها ذخیره شوند.
۲. متناسب با وقفه رخ داده عملیات مورد نظر انجام شود.
۳. ثبات ها مقادیر اولیه خود را باز یابند.
۴. امکان وقفه دوباره فعال شود.
۵. به برنامه اصلی برگردد.

# مثال تابع وقفه

ZRO,	—	/Return address stored here
	BUN SRV	/Branch to service routine
	CLA	/Portion of running program
	ION	/Turn on interrupt facility
	LDA X	
	ADD Y	/Interrupt occurs here
	STA Z	/Program returns here after interrupt
	.	
	.	
		/Interrupt service routine
SRV,	STA SAC	/Store content of AC
	CIR	/Move E into AC(1)
	STA SE	/Store content of E
	SKI	/Check input flag
	BUN NXT	/Flag is off, check next flag
	INP	/Flag is on, input character
	OUT	/Print character
	STA PT1 I	/Store it in input buffer
	ISZ PT1	/Increment input pointer
NXT,	SKO	/Check output flag
	BUN EXT	/Flag is off, exit
	LDA PT2 I	/Load character from output buffer
	OUT	/Output character
	ISZ PT2	/Increment output pointer
EXT,	LDA SE	/Restore value of AC(1)
	CIL	/Shift it to E
	LDA SAC	/Restore content of AC
	ION	/Turn interrupt on
	BUN ZRO I	/Return to running program
SAC,	—	/AC is stored here
SE,	—	/E is stored here
PT1,	—	/Pointer of input buffer
PT2,	—	/Pointer of output buffer