Design of Mechatronic Systems Professor Prasanna S. Gandhi Department of Mechanical Engineering Indian Institute of Technology, Bombay Lecture 19 Microcontroller Programming Philosophy

(Refer Slide Time: 00:11)



In today's class, we will start continuing further our discussion on microcontroller architecture and programming details. So, let's begin with the programming philosophy as we discussed in the last class also, this philosophy would center around registers.

(Refer Slide Time: 00:49)



So, we have seen there are different levels of programming and each one has like different ways to access different registers in the microcontroller platform. And these registers are basically at their fundamental level, they are flip flops. So, you can see hardware has these different layers, one can understand the hardware in different layers, or and one can understand the software also will have like as we will see different levels of programming.

So, the base level is the assembly language where you can access each of the registers by directly assembly language. And we will see how you do that with some examples of some of the hardware of basic fundamental microprocessor 8085 system. And then, we will move on to like the more advanced system and see some examples of programming there and we will move to further advanced system, which is on ARM technology.

We have seen in the last class, general philosophical description or motivation, for which ARM technology was developed. And it has a lot of usages in the modern mechatronic systems. So, that is why I like it is good to study ARM kind of technology and its applications or its interfaces that it provides and things like that.

So, this way you will be able to develop your own mechatronic products in the future, if need be, very easy. And you know what is the state of the art that is happening. So, that is why we will work with this Tiva microcontroller and the Tiva microcontroller will give us lot of understanding about ARM technology.



(Refer Slide Time: 03:09)

Now, we will move on to little bit more details about programming. So, if you go to basic fundamental assembly level language, say 8085 system, the programming would have this something called opcode which is the instruction for, as in C you have some instructions like

that, you will have some instructions in the assembly level programming, which is given in some kind of MOV, ADD these are very short instructions are there.

This MOV as the name suggests it moves contents of register A to registers C, these are, A and C as we saw are registers in architecture of 8085 system somewhere. So, A is a special register called accumulator in that system. So, these are descriptions of these all things will be given in the datasheet and even these commands and their meaning will be given in the instruction manual for the microcontroller or microprocessor in this case.

So, this is most fundamental level programming that one can do, where you actually write assembly language code, nobody does that these days actually. It is just for the sake of academic understanding that, oh look finally whenever we assemble our C code, it finally gets into this program and this program gets executed in its own way.

So, finally, each of these commands also will have some hex code associated with that. So, that is how that code will be loaded into the, into their respective registers. And then based on the timing and control circuitry that one-by-one lines of code will get executed and that is how likely your operation happens. So, this goes for fundamental level understanding of how things happen at a hardware level.

Say, for example, when I say moves contents of register A to register C, there are some hardware pins to be enabled and then needed to made high. So, that contents of the register A are available on the data bus. And then, you enable some other pins so that the contents from the data bus are transferred to register C, these are micro instructions, which will be associated with each of these so-called opcodes.

And number of these micro instructions will determine how much time would take, how many clock cycles in particular would be required to execute this command MOV here. So, that is understanding here so for any command.

(Refer Slide Time: 06:02)



So, in the instruction set you can observe now that there aremany different instructions that will be given. So, for example, this is instruction ACI. So, description is add immediate to accumulator with carry. This accumulator is this register A. So, if you have no understanding of architecture, then these commands will not make much sense.

But, if you have understanding that look this is accumulator and this command as this opcode and operand and then number of times state it takes is 7, or t-states cycles, number of cycles of the clock it will take to execute this command is 7 here. Like that this, all these details will be given and this instruction has hex code this CE 57.

So, olden days, you do not have this, the computer or keyboard to program these 8085 systems. So, what you do is, you will have this ADD display in which there are only like 8

fields. So, 4 are, 6 are, there are only 6 fields, there are 4 of them are address field, and 2 of them will be data field. So, you start at any address and you start coding these hex codes into these different memory locations. And that is how you start writing the program.

So, which was quite cumbersome to create right now, we have done that in our microprocessor course or lab that I was when I was MTech. student here, I did that in one of the labs actually. So, it is quite a time-consuming process. Also, in my B tech. project I did that actually. So, it is very interesting, now if I think about how this, so much of a drudgery to code this and then make sure that they execute properly and run your motors or any other applications.

(Refer Slide Time: 08:34)

Norme Total Appended #0005.pdf Normal Norma Total Appended #0005.pdf Common Normal Total Appended #0005.pdf Total Appended #0005.pdf <thtotal #0005.pdf<="" appended="" th=""> <thtota< th=""><th></th><th>-</th><th></th><th></th><th></th><th></th><th></th><th></th><th>0</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>- 02</th><th></th><th></th><th></th></thtota<></thtotal>		-							0								- 02				
Image Image <t< th=""><th>lome</th><th>Tools</th><th>1</th><th>ppendi-F8</th><th>085.pdf</th><th>×</th><th>MC95</th><th>S12XEP</th><th>100P</th><th>Tiva_Fu</th><th>nctions_S</th><th>TM</th><th>IC123G</th><th>_Laun</th><th></th><th></th><th>0</th><th>#</th><th>Sig</th><th>n In</th></t<>	lome	Tools	1	ppendi-F8	085.pdf	×	MC95	S12XEP	100P	Tiva_Fu	nctions_S	TM	IC123G	_Laun			0	#	Sig	n In	
000000011 A_X Little 'The conditional Subroutine Call Opcode Operand Bytes MClyttle' Tobal Subroutine Call Opcode Operand Bytes Mclyttle subscience Call Subroutine Call Description The program supersce in transformed to the address specified by the program country in public on the stack. The supersce of events in the subroutine location 200HI to call a subroutine location 200HI to call a subroutine location 200HI. Count to 'Monone Wood' Supersce'' Memory Hirk Memory Elick Memory Elick Memory Elick Count to 'Monone Wood' Supersce'' 2010 CD CALL 200HI Count to 'Monone Wood' Supersce'' Mcmory Elick Supersce'' Count to 'Monone Wood' Supersce'' 2011 CO Marce 100 Minone Marce		P	θ	Q 🛈	٩	7	/ 47	k		€ €	88.2% *	B .		Ģ	l	Ó.	9	Ø.		1	
CALL Doconditional Subroutine Call Opcode Operand Bytes M-Cycles 7-States Hex Code CALL 5648 3 5 H CD Description The program concers in sandarded to the address specified by the operand Botte the transmit, the address of the next instruction to CALL the contents of the program control is publed on the state. The sequence of events is discribed in the categories before. Plags No flags are affected. Description The program control is publed on the state. Due sequence of events is discribed in the categories before. Plags No flags are affected. Description The requests of events when the state jointer is at location 2010 CD CALL 2050H 2011 2 20 Note: See the difference between when the state jointer is at location 2010 CD CALL 2050H 2012 2 20 Note: See the difference between when the high operative (20) is strend. The code, the low-order by the (3) is earded for the high operative (20) is strend. CC Call with No Carry CY = 1 DC CYC Call with No Carry CY = 1 D	ę		_		1000	000	11	A	83	110	,1,0,10 P		<u>1</u>				Search 'C	ombine	PDF		
Open of plan open open open open open open open ope	1		CALL	Uncond	itiona	Subr Rytes	outine M-6	e Call	T-States	Hex Co	ode				1		D. Even	out DDE		~	
Addems Description The program sequence is standarding to the address specified by the program counter) is public on the stack. The sequence of events is described in the cannels. Before the market, the address at the sequence of events is described in the cannel at 2000 LC 2011 instruction at memory location 2010 It to call a subrowine lo- cannel at 2000 LC 2010 LC 2010 It 2011 20 2010 LC 2012 and the sequence of events when the stack pointer is at location 2011 20 20 CC 2010 LC 2010 It 2011 20 20 20 20 CC 2012 COUNT 2011 20 20 CC 2010 CC 2012 COUNT 2011 20 20 CC 2010 CC 2012 CC 2010 CC 20		1	CALL	16-b	út	3	14-4	5	18	CD						P	CO Exp	on PUP		~	
Description The program respects is transfored to the address specified by the operative characteris of the address is a classified to CLU the content of the program content) is pushed on the stack. This sequence of events is described in the cample blow. Fags No fags are affected. Example Write CLU instruction at memory location 2010ff to call a subtroatine location at 2010ff. Explain the sequence of events when the stack pointer is at location 2019ff. Memory Her Machine CLU instruction at memory location 2010ff to call a subtroatine location 2010ff. 2010 CD CALL 2000ff. 2010 202 30 New: See the difference between switting a field address as amemonies and code. In the code, the low-control to (3) is extend. For the code, the low-control to (3) is extend. CC Call on Carly CD is low for the the memories and code. In the code, the low-control to (3) is extend. CC Call on Carly CD is call address as a memories and code. In the code, the low-control to (3) is extend. CC Call on Carly CD is call address as a memories and code. In the code, the low-control to (3) is extend. CC Call on Carly CD is control of the mode memories of the manufactor of the common the field address is a second to be stated on the state point is a second to be stated to call a subtroation in the time is an analysis. CC Call on Carly CD is control of the control by CD is extend. CC Call on Carly CD is control of the control by CD is extend. CC Call on Carly CD is control of the control by CD is extend. CC Call on Carry CD is control of the control by CD is extend. CC Call on Carly CD is control of the control by CD is extend. CC Call on Carly CD is control of the control by CD is extend. CC Call on Carly CD is the field of the field does in the text contines. CC Call on Carly CD is control to the control by CD is extend. CC Call on Carly CD is the field does in the text contines. CC Call on Carly CD is the field does in the call is the streak points more Tatates. CC Call on Party Extend. Figs: No figs a	2	1		addre	155											A	dobe Exp	ort PDF		Ф	
the program control is pruced on the strick. The sequence of events is described in the example below. Figs: No flags are affected. Reample Write CALL instruction at memory location 2010H to call a subroutine located at 2000L Explain the sequence of events when the stack pointer is at location 2009H. Convert to Memory File: Address Code Misemonics. 2010 CD CALL 2050H 2010 CD 2011 50 CD CALL 2050H Convert Language: 2010 CD CALL 2050H Convert Language: Convert Language: 2010 CD Convert Language: Convert Language: Convert Language: 2010 CD Convert Language: Convert Language: Convert Language: 2011 Solution to the converter language: Convert Language: Convert Language: Convert Language: 2011 Convert POF Files Modes to co			Descrip	tion The Before the	program transfe	n seque r, the ad	ince is idress o	transfer f the net	rred to the ct instruction	address sp to CALL (ecified by the the contents of					Ci or	envert PDF Excel Onli	Files to ine	Word		
Flags No flags are affected. Example Write CALL instruction at memory location 2010H to call a subtoutine location 2009H. Convert to Memory Hex Address Code Minemodes 2010 CD CALL 2009H. 2010 CD CALL 2009H. Memory Hex Address a memonics and code. In the code, the low-order byte (20) is metred first, then the high-memories and code. In the code, the low-order byte (20) is metred first, then the high-memories to commente it is an analyzed first. Convert to Note: See the difference between writing a 16-bit address as memonics and code. In the code, the low-order byte (20) is metred first, then the high-memories to commente it is metred. Convert to the low order byte (20) is metred first, then the high down and the memory low order byte (20) is metred. IPTEL CC Call on Chr.y CY = 1 Convert 10 Memory CY = 0 CY = 1 Convert 10 Memory CY = 0 Convert 10 Memory CY = 1 Convert 10			the pro	gram counte below.	rt) is pu	shed on	the sta	ck. The	sequence of	events is o	escribed in the					84	Nect PDF I	File			
Example Write CALL instruction at memory location 2016ff to call a subcostine location 2009ff. Convert 10 Memory Hex Address Code Mnemonics 2010 CD CALL 2050ff. 2010 CD CALL 2050ff. CD CD CD 2010 CD CALL 2050ff. CD			Flags	No flags ar	e affecte	rd.										. '	Appendi-Fi	8085.pd		×	
eard in 20001. Explain the sequence of events when the stack pointer is at location 2009H. Memory Hex Address Code Macmonics 2010 CD CALL 20001 2011 20 2012 20 Note: See the difference between writing a 16-bit address as macmonics and code. In the code, the low-order byte (30) is entered first, then the high-order byte (20) is entered. If a supermitted in the sequence of the second byte (20) is entered. If a supermitted in the sequence of the second byte (20) is entered. If a supermitted in the sequence of the second byte (20) is entered. If a supermitted is the sequence of the second byte (20) is entered. If a supermitted is the sequence of the second byte (20) is entered. If a supermitted is the second byte (20) is entered.		3	Examp	le Write (CALL in	structio	n at m	emory k	cation 2010	H to call a	subroutine lo-					C	onvert to				
Memory Her Address Code Memories 2010 CD CALL 2050H 2010			cated a 2099H.	2050H. E	xplain t	he sequ	ence of	f events	when the st	ack pointer	is at location					De	dicrosoft v	Mord (*.d	(CCK)	~	
Address Code Minimotocs 2010 CD CLL 2050II 2011 50 2012 20 Note: See the difference between writing a 16-bit address as memonoics and code, In the code, the low-entropic (20) is mattered. the code, the low-entropic (20) is mattered. PTEL One CC CL If the difference between writing a 16-bit address as memonoics and code, In the code, the low-entropy (20) is mattered. TPTEL One CC CL If the difference between writing a 16-bit address as memonoics and code. In the code, the low-entropy (20) is mattered. TPTEL CC Call on pairly CM CC Call on Carry CY = 0 DC CC Call on Carry CY = 0 D4 CO Assert How Readition is to true CN CL If the Pine DE CP					N	lemory	He	ex													
2011 30 2012 20 Note: See the difference between writing a 16-bit address as mnemonics and code. In the code, the low-ender bit (30) is metred. True Online 1000 Support True True True Online 1000 Support True, Field Support Colspan= Support <td co<="" td=""><td></td><td>콜</td><td></td><td></td><td>A</td><td>adress 2010</td><td>0</td><td>ide D (</td><td>CALL 2050E</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Create,</td><td>edit and</td><td>sign PD ments</td><td>ŧ.</td></td>	<td></td> <td>콜</td> <td></td> <td></td> <td>A</td> <td>adress 2010</td> <td>0</td> <td>ide D (</td> <td>CALL 2050E</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>Create,</td> <td>edit and</td> <td>sign PD ments</td> <td>ŧ.</td>		콜			A	adress 2010	0	ide D (CALL 2050E								Create,	edit and	sign PD ments	ŧ.
2012 20 Note: See the difference between writing a 16-bit address as mamonoiss and code, In the code, the low-ender byte (20) is entered. Itext Tree Training and the code of the code of the code of the code of the low-ender byte (20) is entered. PFTEL Itext Tree Training and the code of the c		-				2011	5	0	CALLS SOLVE								- Arrite				
Office Convert 10 Convert 10 More 1 Convert 10 Convert 10 Convert 10 More 1 Convert 10 More 10 Convert 10 More 200 <																					
Image: Construction C	NPTEL								A	Appendi-F8	8086.pdf			_			0				
CC Call on City CY = 0 DC 20 (C condition is not true) CNC Call with No Carry CY = 0 D4 5/18 of C condition is not true) CNC Call on minus S = 1 PC S/18 of C condition is not true) CP Call on minus S = 1 PC rewer T states. CP Call on Parity Den P = 0 E free ordition is true is continues CP Call on Parity Den P = 0 E free ordition is true is call the scalars. CNZ Call on Zero Z = 1 CC submotine, thus requires more CNZ Call on No Zero Z = 0 C4 T states. CNZ Call on No Zero Z = 0 C4 T states. CMA: Complement Accumulator Opcode Operand Bytes M-Cycles Opcode Operand Bytes M-Cycles T states Intraction CMA: Complement Accumulator Opcode Operand Bytes M-Cycles T states Description The contents of the accumulator, which has data byte 89H. Intraction: Conset. edit and sign POF Conset. edit and sign POF Moreast and sign POF Editore instruction After instruction After instruction Stater instruction <td>NPTEL ome</td> <td>Tools</td> <td></td> <td>ppendi-F8</td> <td>085.pdf</td> <td>х</td> <td>MC95</td> <td>612XEP</td> <td>(00P</td> <td>Appendi-F8 Tiva_Fu</td> <td>8086.pdf inctions_S</td> <td>TM</td> <td>IC123G</td> <td>Laun</td> <td></td> <td></td> <td>()</td> <td></td> <td>Sig</td> <td>n In</td>	NPTEL ome	Tools		ppendi-F8	085.pdf	х	MC95	612XEP	(00P	Appendi-F8 Tiva_Fu	8086.pdf inctions_S	TM	IC123G	Laun			()		Sig	n In	
CNC Call on points S = 1 PA Most Provide on the sequence, and thus requires CP Call on minis S = 1 PC the sequence, and thus requires CP Call on pairly Vold P = 0 EA the sequence, and thus requires CP Call on pairly Vold P = 0 EA the sequence, and thus requires CP Call on Pairly Vold P = 0 EA If condition is true it calls the sambrourine, this requires more CNZ Call on No Zero Z = 1 CC submotine, this requires more CNZ Call on No Zero Z = 1 CC submotine, this requires more CNZ Call on No Zero Z = 1 CA Testates. Paigs No flags are affected. Adobe Export PDF File Adobe Convert FDF File Opcode Operand Bytes M-Cycles T-States Hex Cade CNA None 1 4 2F Description Moreout Worl ("doce) v Document Language: Description The contents of the accumulator, which has data byte 89H. Instruction: After instruction Before instruction After instruction <t< td=""><td>NPTEL</td><td>Tools</td><td></td><td>ppendi-F8i 그 ①</td><td>085.pdf</td><td>×</td><td>MC95 / 47</td><td>S12XEP</td><td>100P</td><td>Appendi-F8 Tiva_Fu -) (+)</td><td>8085.pdf nctions_S 88.2% *</td><td>TM-</td><td>IC123G</td><td>Laun</td><td>Q</td><td>¢.</td><td>0</td><td>\$ 8_</td><td>Sig</td><td>n In</td></t<>	NPTEL	Tools		ppendi-F8i 그 ①	085.pdf	×	MC95 / 47	S12XEP	100P	Appendi-F8 Tiva_Fu -) (+)	8085.pdf nctions_S 88.2% *	TM-	IC123G	Laun	Q	¢.	0	\$ 8_	Sig	n In	
CM Call on minus S = 1 FC the sequence, and thus requires CPE Call on Pairty Even P = 1 EC fever T-states. CPO Call on Pairty Odd P = 0 E4 If condition is true it calls the STACE CONC Call on Pairty Odd P = 0 E4 If condition is true it calls the STACE CALL on Pairty Odd P = 0 E4 If condition is true it calls the States CONC Call on No Zero Z = 1 CC submoutine, this requires more CNZ Call on No Zero Z = 0 C4 T-states. Flags No flags are affected. CMA: Complement Accumulator Opcode Operand Bytes M-Cycles T-States Hex Cade CMA None 1 1 4 2F Description The contents of the accumulator are complemented. Flags No flags are affected. Canset, est and sign POF from & a greanments Instruction: CMA Hex Code: 2F Before instruction Ahre instruction Meet Code: 2F	ome	Tools		ppendi-F8i 그 ① Call on Cat	D85.pdf	* 8 C	MC98 / 47	S12XEP	100P	Appendi-Fi Tiva_Fu 	8085.pdf nctions_S 88.2% * K# X*Otatica ition is not true)		ic123G	_Laun	l.	ć.	() D Bearch 'C		Sig	n In	
CPO Call on Parity Odd P = 0 E4 If condition is true it calls the canon convert PDF File Adobe Export PDF Convert TDF Convert to Convert to Conve	ome	Tools		ppendi-F8 Call on Car Call with N Call on pos	D85.pdf	× 8 C	MC95 / 47 (' = 1 (' = 0 S = 0	S12XEP	(000P)	Appendi-F8 Tiva_Fu 	8085.pdf inctions_S 88.2% * 88.2% * ition is not true) dition is not true) is not true it con	TM-	iC123G	Laun	l.	¢.	() D Search 'C	A Ca	Sig Sig	n In	
CNZ Call on No Zero Z = 0 C4 Testates. Plags No flags are affected. Select POF File Append-FileS to Word or Excel Online CMA: Commet POF Files to Word Select POF File CMA: Commet POF Files Append-FileS to Word CMA: Commutator Append-FileS to Word Opcode Operand Bytes M-Cycles CMA: None 1 1 4 Description The contents of the accumulator are complemented. Moreosch Word (*docn) ∨ Description The contents of the accumulator, which has data byte S9H. Convert to Instruction: CMA Hex Code: 2F Before instruction After instruction	orme	Tools	C NC P M PE	ppendi-F80 Call on Car Call on Car Call on pos Call on mir Call on Par	D85.pdf () TY' Io Carry itive nui ity Even	8 C	MC98 / 47 t' = 1 t' = 0 S = 0 S = 1 P = 1	DC D4 F4 FC EC	(A) 1000P 2 5 Note: 11	Appendi-Fi Tiva_Fu) (f condi /18 (if condi /18 (if condi condition the sequen fewer T-st.	8086.pdf inctions_S 88.2% * sex *Orasco ition is not true) is not true it con tee, and thus req ates.	TM-	IC123G	Laun	2	¢.	Image: Search 'C Imag	A Ca Iombine I ort PDF	Sig 2017		
Flags No flags are affected. Select POF Fie COMA: Complement Accumulator Appendi-F0005.pcf Opcode Operand Bytes M-Cycles T-States Hex Code CMA None Instruction: The contents of the accumulator are complemented. Flags No flags are affected. Example Complement the accumulator, which has data byte 89H. Instruction: CMA Effore instruction After instruction	ome	Tools	C NC P M PE PO 7	ppendi-F8 Call on Car Call on Car Call on Par Call on Par Call on Par Call on Par	D85.pdf TY' lo Carry itive susi ity Event ity Event	8 C C	MC95 / 47 f = 1 f = 0 S = 0 S = 1 P = 1 P = 0 Z = 1	DC DC D4 F4 F4 FC EC	(100P)	Appendi-Fi Tiva_Fu) (if condi /18 (if condi /18 (if condi condition i fewer T-st condition i	8085.pdf 88.2% = 88.2% = 88.2% = 88.2% = 88.2% = 80.2% = 80	TM-	iC123G	_Laun	L.	¢.	© Search 'C C Expe	A Canonical Control PDF	Sig Sig		
CMA: Complement Accumulator Opcode Operand Bytes M-Cycles T-States Hex Code CMA Nore 1 1 4 2P Description The contents of the accumulator are complemented. Flags No flags are affected. Example Complement the accumulator, which has data byte 8991. Instruction: CMA Hex Code: 2F Before instruction After instruction Before instruction CMA Hex Code: 2F	Iome	Tools	CC NC P M PE PO Z NZ	ppendi-F8 Call on Car Call on Car Call on Par Call on Par Call on Par Call on Par Call on No	D85.pdf TY Ty Ty To Carry tive twi ty V Co Carry ty Co Carry Co Carry Co Carry Co Carry Co Carry Co Co Co Co Co Co Co Co Co Co	8 CC CC	MC95 7 47 47 = 1 5 = 0 5 = 1 7 = 1 7 = 0 7 =	DC DC D4 F4 FC EC E4 CC C4	(A) 1000P 2 5 Note: It 11	Appendi-Fi Tiva_Fu Tiva_Fu (f condition) (f condition) (f condition) (conditio	10085.pdf nctions_S 88.2% * 1000 is not true) 1000 is not true is constant 1000 is not true is co	TM- tinues uires e nore	uc123G	_Laun	2	Ć. Al	Search 'C Search 'C C C C C C C C C C C C C C C C C C C	Ort PDF ort PDF Files to ine	Sig POP Word	n In	
Opcode Operand Bytes M-Cycles T-States Hex Cade Convert to CMA Nore 1 4 2F Moreout Worl ("doot) ~ Document Language: Description The contents of the accumulator are complemented. Document Language: Document Language: Flags No flags are affected. Complement the accumulator, which has data byte 8991. Consets, soft and sign POF Instruction: CMA Hex Code: 2F Before instruction After instruction Before instruc	NPTEL	Tools	CC CC CC CP M M SPE Z Z Z Z Z	ppendi-F80 Call on Car Call on Car Call on Par Call on Par Call on Par Call on Par Call on Par Call on No Flag	D85.pdf (1) (2) TY (0) Carry tive ty Odd Od Zero (3) No (1) (3) (4) (4) (4) (4) (4) (4) (4) (4	8 CC CC	MC95 i' = 1 i' = 0 S = 0 S = 1 P = 1 P = 0 Z = 1 Z = 0 affecte	DC DC D4 F4 FC E4 CC CC C4	(000P)	Appendi-Fi Tiva_Fu	8086.pdf nections_S 88.2% = 88.2% = structures tition is not true) tition is not true it tition is true it cells the task. thus requires m	tinues uires e nore	HC123G	_Laun	2	Ó. Ad Ci	Search 'C Search 'C C Search	Order O	Sig PDP Word	n In	
CMA None 1 4 2F Moreooft Word (* docs) ∨ Description The contents of the accumulator are complemented. Document Language: Flags No flags are affected. Example Complement the accumulator, which has data byte 8991. Instruction: CMA Here Science and Regional accumulator, which has data byte 8991. Before Instruction:	IPTEL	Tools	CC CNC PP CM PPE PPO ZZ ZZ ZZ	ppendi-F80 Call on Car Call on pos Call on pos Call on par Call on Par Call on Par Call on So Flag	D85.pdf TY TY tive ty Odd o Zero s No 1 TA: Co	× 8 CC CC	MC9S / 47 f = 1 f = 0 S = 0 S = 1 P = 1 P = 0 Z = 1 Z = 0 affecte	DC DC D4 F4 FC E4 CC C4 dd.	Note: If	Appendi-Fi Tiva_Fu	80.86.pdf netions_S 80.2% • sa romana tition is not true) difficon is true) is not true it core cce, and thus requires no states. is true it calls th , thus requires no	TM-	UC123G	_Laun	2		Search 'C Search 'C Expire Search 'C Expire Search 'C Expire Search 'C FIF	Ornt PDF Files to res	Sig POP Word	n In	
Description The contents of the accumulator are complemented. Document Language: Flags No flags are affected. Create, edit and sign POF Example Complement the accumulator, which has data byte 89H. Create, edit and sign POF Instruction: CMA Hex Code: 2F Before instruction After instruction Before instruction	oome	Tools () () () () () () () () () ()	The second secon	ppendi-F8 Call on Car Call on Par Call on pos Call on par Call on Par Call on Zer Call on No Flag CM Opc	D85.pdf (1) Ty' lo Carry tive lo Carry tive tive tive tive to Carry tive to Carry tive tive to Carry tive tive to Carry tive tive tive tive to Carry tive tive to Carry to Carry t	* 8 CC CC	MC9S / 47 t' = 1 t' = 0 S = 0 S = 1 P = 0 Z = 1 Z = 0 affecte nent J ad	B12XEPP DC D4 F4 F4 FC EC EC E4 CC C4 Fd.	Note: It uulator M-Cycles	Appendi-Fi Tiva_Fu Tiva_Fu Tiva_Fu Tiva_Fu Tiva_Fu T-states T-Stat	8086.pdf nections_S 88.2% = x-y-sase tion is not true; to so not true; to thus requires m tes: Hex Coc	The	HC123G		2		Search 'C Search 'C Experience POF I Excel Onli Idobe Exp Appendi-Fi	Ort PDF Files to ine File 8005.pd	Sig POP Word	n In 1	
Flags No flags are affected. Onate, edit and sign POF Example Complement the accumulator, which has data byte 89H. Onate, edit and sign POF Instruction: CMA Hex Code: 2F Before instruction After instruction Before instruction		Tools () () () () () () () () () ()	CC NC P P M PP6 Z NZ	ppendi-F88 Call on Carl Call on Carl Call on Para Call on Para Call on Para Call on Para Call on Para Call on Para Call on Carl Call On Carl	D85.pdf TY TY Ty to Carry tive two to Carry ty Odd o Zero s No I (A: Co. ode IA	* 8 CCCC CC	MC95 / 47 (= 1 (= 0 S = 0 S = 1 P = 1 P = 0 Z = 1 Z = 0 affecte nent J ad	DC DC D4 F4 FC EC EC E4 CC C4 Accum Bytes 1	Note: If Mulator M-Cycles 1	Appendi-FI Tiva_Fu () () () () () () () () () () () () () (8086.pdf nections_S 88.2% - stores rouses tion is not true; to is not true; to is not true; to is not true; to is not true; to thus requires m ats. tes Hex Coc 2F	TM-	₩ ₩	Laun	2		Search 'C Search 'C Experience of the search 'C Experience of the search 'C Excert Online Appendi-Fit Appendi-Fit anvert to	Ca Combine I Ca Combine I PDF Files to ine B8085.pd	Sig DOP Word f locx)	n In A X X	
Example Complement the accumulator, which has data byte 89H. Create, est and sign POP torms & agreements Instruction: CMA Hex Code: 2F Before instruction After instruction Before instruction Court Press Trial	NPTEL	Tools	C C C C C C C C C C C C C C C C C C C	ppendi-F80 Call on Cat Call on Call on Cat Call on Par Call on Par Call on Par Call on Par Call on Par Flag Opc CM Opc CM	D85.pdf (1) TY Io Carry itive uui ity Even ity Od Zero Zero Zero X: Co. ode IA Co. ription	* 8 CC CC CC Mallags are mplen Opera None The c	MC95 7 47 f = 1 f = 0 S = 0 S = 1 P = 0 Z = 1 Z = 0 affecte nent <i>i</i> and	S12XEP DC D4 F4 F4 FC EC CC CC C4 Accum Bytes 1 of the s	Note: If Mulator M-Cycles 1 ccumulator a	Appendi-FI Tiva_Fu (P) (I conditional (P) (I conditional (P) (I conditional (P) (I conditional (P) (I conditional (P)	8086.pdf nctiona_S 88.2% * sr - vorance lifon is not true; difon is not true; difon is not true; is not true it coc coc, and thus requires a tes true it calls th , thus requires a tes Hex Coc 2F nented.	TM-	ic123G	Laun	0		Expendence Expendence Expendence Expendence Appendence Appendence Appendence Appendence Comparison C	Combine Combi	Sig POP Word f		
Instruction: CMA Hex Code: 2F Before instruction After instruction But True True	NPTEL	Tools	CC NC PP M PP PP PP O Z Z NZ	ppendi-F8 Call on Call on Call Call with N Call on Par Call on Par Call on Par Call on Par Call on No Flag CM Opc CM Desc Flag	D85.pdf () TY fo Carry five Even ity Even ity Odd o Zero Zero S No 1 (A Tription s No f	* 8 CCCC CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC	MC9S $i \neq 47$ $i \neq 1$ $i \neq 0$ S = 0 S = 1 P = 1 P = 0 Z = 1 Z = 0 affecte ment <i>i</i> and affectes	DC DC D4 FC EC E4 CC C4 ad. Accum Bytes 1 of the a d.	Note: It M-Cycles L L L L L L L L L L L L L L L L L L L	Appendi-FF Twa_Fu (4) (4) (6) (6) (4) (6) (7) (4) (4) (6) (7) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4)	8086.pdf nctiona_S 86.2% * sr-vorace liion is not true; dition is not true; ition is not true; ition is not true; it rue it calls to rue it calls to rue it calls is rue it calls is true it calls	The	¢C123G	Laun	0		Search 'C Expendional Control of Contro of Control of Control of Control of Control of Con	Combine I ort PDF ort PDF File 8085.pdl Word (*.d anguage	Sig POP Word f		
Before instruction After instruction Start Free Trial			C NC P P P P Z Z NZ	ppendi-F88 Call on Call on Call on Call on Call on Call on Call on Par Call on Par Call on Par Call on Par Call on Par Call on Par Call on Cal	Dass.pdf Ty	* 8 CC CC CC Hags are mploz Opera None The c lags are	MC9S i' = 1 i' = 0 S = 0 P = 1 P = 0 Z = 1 Z = 0 affecte nent <i>i</i> affecte ment the	DC D4 F4 F4 FC EC E4 CC C4 Accum Bytes I of the a d.	Point Contract of Contrac	Appendi-FF Twa_Fu	10085.pdf netions_S 80.294 * res 1*20405 ition is not true) is not true it core cc, and thus req ates. is true it calls th , thus requires n tes Hex Core 2F nented. yte 89H.	TM4	uc123G	Laun	2		Search 'C Search 'C Search 'C Expel Constant Search 'C Constant Create, tormate,	Ornt PDF Files to rort PDF Files to anguage edit and a & agree	Sig POP Word f locx) c sign PC		
		Tools	CC CC CC CC CC CC CC CC CC CC CC CC CC	ppendi-F80 Call on Call with N Call on Call on Mar Call on Par Call on Par Call on Par Call on Par Flag CM Oppe CM Desc Flag Exar Instr	D85.pdf Ty Ty lo Carry lo Carry itive sus ity Odd o Zero s No I A ription s No f nple (consecution: consecution:	* 8 CC CC CC CC CC CC CC CC CC CC CC CC CC	MC98 7 47 r = 1 r = 0 r = 1 P = 0 r = 1 P = 0 r = 1 P = 0 r = 1 r = 0 r = 1 r = 0 r = 0 r = 1 r = 0 r = 0 r = 1 r = 0 r = 0 r = 0 r = 1 r = 0 r = 0	s12XEP DC DC DC DC EC EC EC E4 CC C4 d. Account Bytes 1 of the a d. e accumi	A constraints of the second se	Appendi-FI Twa_Fu 	8085.pdf netions_S 80.234 - se rotatos tion is not true) diton is true) is not true it conce, and thus requires n tes. Is true it calls th thus requires n tes. Hex Cot 2F nented. yte 89H.	TM4	UC123G	Laun	2		Search /C Search /C Expended	Combine Combi	Sig PDP Word (oocx) c sign PC ments		

So, it will have all the instructions that are available for 8085 system to go, they will be all there in this. So, there are some conditional instructions, unconditional call will be there. There will be some if-else statements how do they execute that you will get to know say for column positive. So, if something is positive, then you will shift the address, next address which will be from where the programming execution will restart. Like that you will have some kind of nesting loops possibility and things like that, by using this basic fundamental level commands.

So, nowadays, all modern microcontrollers you just say if-else in C language and your job will be done. But in olden days there is no command if you will find in this whole set of instructions for this, there is no command- if or there is no command- while. So, there are these commands called call on positive, call on carry or call on 0. So, that means when the flag, you have seen the flag we studied in the last class, when the 0 flag and that flag register becomes 1, then the program some sub-routing will be called for the program and that is how this instruction works actually, call.

So, this is how olden days this is programming used to be, and that is what is done now also, but it is all the compilers to do that targery. So, human beings are spared from that. So, we as a programmers for hardware programming write all the codes in C language and they get executed. So, that is how things happen there.

So, main understanding here is that this 8085 system or any microcontroller system will ultimately have the program in the assembly language that will be downloaded to microcontroller and each of these instructions of course, they will have associated some hex code with that and given higher level program that you will write in C, the assembler will, or compiler will compile this program and generate these opcodes and then generate a code, which will be only with respect to these opcodes and that is a code which will be downloaded microcontroller memory that is how the things are going to happen.

And you will see later when we see the interface of Tiva microcontroller, we will be able to see that some of these opcodes or some of the assembly level parts of the code which are there in the microcontroller.

(Refer Slide Time: 11:55)



These are fundamental operations that will happen in microcontroller at the assembly level. So, all these operations are that way universal I would say, but one can see there are some opcodes or instruction sets which are in the assembly level for each of these fundamental operations. There are some logical operations, there are arithmetic operations, we saw this command ADD, then internal data transfer, we saw this command MOV.

So, like that there are this branching operation there this call command or jump or jump on carry, jump on 0, these are commands, which are for branching. So, you are executing some series, the commands, and from here now based on some condition, I want to execute something else, I will jump from there to something else execute that and come back and keep doing that. That way, you will have a facility to create loops and you have facility to create if-else statements and things like that as I said earlier.

So, these are the ways in which things actually happen at the assembly level. So, we do not need to get into whole lot of details here. But we need to have the sense that look, whatever commands I am writing here in the higher-level language, they will have, compiler will actually compile them and convert them into these opcodes. And then, those opcodes will be finally executed automatically right now.

(Refer Slide Time: 13:38)



Now, let us move further for little bit higher level microcontroller. So, if you see your Arduino microcontroller or your XEP 100 their philosophy is (I mean philosophy) is the same you need to have a finally convert those codes into the assembly level language. But to access the registers, you do not have these register names A, B, C, D as you had in the 8085 kinds of a system, you will have these registers access there in a different way.

So, these registers are available as a variables in C, some variable in C. These variables would be defined in the datasheet of the microcontroller. And also, they will have header files provided to you in which those registers are mapped into, those variables are mapped into certain memory locations or registers finally. So, when you write to that variable, it will go to appropriate register in the microcontroller system, that data value whatever you have written in that variable, it will go to that particular register.

And there are two types of such registers, one is control register, and other is working register. Control registers for any interface, there will be these two kinds of registers. So, we will see some examples. Say for example, maybe I will explain first what this function is and then we will take a example.

So, control registers set up your interface for particular or configures your interface for a particular operation to happen and working register actually supply the working values which will be continuously changing.

So, control registers you set up only once in your program and then working register you may keep on changing based on whatever closed loop operation that you are carrying out. For example, if you remember or some of you might have already done this programming of XEP 100 in that this DDRA register was there, DDRA variable was defined.

So, the call is DDRA now register only although it is a variable in C you would because this is mapped to a certain register which is fixed in the computer or a microcontroller memory, it can be referred to as DDRA register. So, say this register DDRA in digital input output interface in XEP 100 it controls as the name suggests direction definition register.

So, DDR is direction definition register. So, it defines the direction for the data transfer in this register pins of register A and this is A port that is available on the hardware pins of this microcontroller. So, this all will be given in the datasheet. We will look at this XEP 100 data sheet also in a minute. But this DDRA that way happens to be a control register then, you do not keep on changing the data directions typically in the register.

So, once you define, I am using this register as an output register and the pins are connected to some output LEDs, I will not change the direction now, once it is set, we will execute that for say some display of some numbers or dancing LEDs or some LED to glow for to indicate some operation happening that is the functionality that will built into the microcontroller program.

So, that way this DDRA register is a control register. Now control register will just set up the things, it will not put values into any of the functions actually. So, for that you need a working register that is working registers will actually start putting values in that and outputting something that you want to execute in your final program.

So, for example, DDRA will set this register to have an output definition for pins of port A and you actually write to this port A some data and then like that data will be displayed if it is output port, that is how you have written this program. So, we will see that now in a datasheet of this microcontroller. So, let us switch to that.

(Refer Slide Time: 18:33)



So, this is XEP 100 reference manual. Now, if you see this manual, it will have 1324 pages. So, we do not get into reading all these manuals to start putting programming. So, we want see this the module that is responsible for actually doing this data input output, is this port integration module and we can directly go to see what is there in that module.

(Refer Slide Time: 19:01)





And you can see that you can read through some of these details as is port A port B something is given then there are these different variables will be given. Then number of pin definitions will be given like that all these things are given. So, you can, if you know this name of some registers, you can look for those registers as there can say control find and look for these registers.

(Refer Slide Time: 19:38)





So I say here DDRA and I get all the details about this register. So, we can see here, so this is data register DDRA data direction registers for port A. So now, it gives me all the definitions if the value is written there, it will be defined as input port or output port like that. It gives all the description and according to what is written in the datasheet if I set up this register some number, then accordingly the functions will start happening in the hardware.

So, that is how one goes about programming these different-different microcontroller interfaces. So, this is about this simple digital input output interface. These are many different you have DDRB, DDRC and things like that. Then you will have port register port A, port B, port C is a data register. So, port A when you say like the data will be given to this port or read from this port.

Port C means the data will be, suppose I want to write or this is port is output port, I will write this output number into this port and depending upon what is your data say that data is 10101010, then wherever that 1 is there that LED will glow, 0 is there LED will be shut off or something like that.

So, you are putting the hardware pin high and low values on the hardware pin by putting these data into respective registers, that is our operation that takes place in the input output ports.

So, this is how we look for programming for little higher-level microcontrollers. So, this thing you will be able to observe for many different kinds of interfaces. So, these different-different modules, so, on these, you see different chapters in the datasheet we will find what are modules that are provided by this microcontroller or in the datasheet you will find all these modules that are provided and more details of those interfaces will be observed in the respective chapters of the datasheet.

(Refer Slide Time: 22:15)



So, typical interfaces will be of these different-different kinds. So, depending upon microcontroller that you choose you will have these different interfaces and interfaces names also will change from one microcontroller to another microcontroller. Say for example, this digital input output interface as we saw just now in XEP it is called port integration module or PIM. In the Tiva, it has a name GPIO or general-purpose input outputs. So, like that you will have different-different names in different microcontroller. So, do not get bogged up with that you can just understand this is your interface that you need to program.

Typically for mechatronic applications from sensors perspective, we will need these interfaces, general purpose digital i/o input outputs. Analog to digital conversion and then quadrature encoder interfaces and maybe some serial or other communication modes can be possibility. Especially if the sensors give directly the output in terms of serial port or port like a CAN interface then we will need these serial or other communication modules to be programmed.

Actuator typically will run by PWM module. In some cases, there is a possibility of running the actuator by using this digital to analog conversion also, but in most of our cases it may not happen.

(Refer Slide Time: 23:55)



So, I think maybe now we will just talk about little bit more details of these interfaces. But we will come back maybe in the in the next class. So that I have this small chunk of thing about just to maybe base fundamentals of these and then we can come back to this more details of these interfaces in the next part of the today's class. So, I will stop here for now.