Program Control Instructions

•Generally Modify CS:IP

•Causes Modification in Execution Sequence (of Instructions)

• When Such a Program Flow Change Occurs: a) Instructions in the BIU Inst. Queue Become Invalid b) BIU Directly Fetches cs:IP Instruction from Memory

c) While EU Executes New Instruction, BIU Flushes/Refills Inst. Queue

Classification

a) Jumps - Unconditional Control Transfers (synchronous)

- b) Branches Conditional Control Transfer
- c) Interrrupts Unconditional Control Transfers (asynchronous) d) Iteration More Complex Type of Branch

Control Instruction Summary

UNCONDITIONAL ;next instruction executed has LABEL ;next instruction executed in a filer the call ;next instruction executed is a filer the call ;nothing executed until RESET signal

jmp LABEL call LABEL ret hlt

int into iret

loop LABEL loope/loopz LABEL loopne/loopnz

> <immed8> <immed8>

ITERATION ;cx <-- cx - 1, jump to LABEL if cx > 0 ;same as loop but ZF=1 also required ;same as loop but ZF=0 also required

INTERRUPTS ;Invoke the int. handler specified by immed8 ;same as int but OF=1 also ;Return from interrupt handler

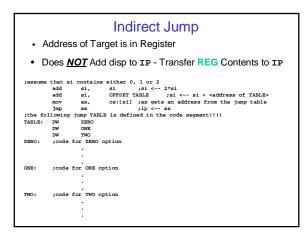
CONDITIONAL to follow

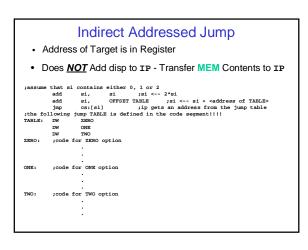
Simplest Control Instruction, jmp LABEL ;LABEL is offset address of instruction jmp ; in the code segment THIS IS EQUIVALENT TO: OFFSET LABEL mov ip, The mov form is illegal since it is not allowed to do this inside the code segment 3 Forms of jmp SHORT - 2 bytes, allows jump to ±127 locations from current address EB disp NEAR - 3 bytes, allows jump to ±32K locations from current address E9 disphi displo FAR - 5 bytes anywhere in memory EA IP lo IP hi CS lo CS hi

LEA [IP to] [IP hi] [CS to] [CS hi] 386+ - Must Far Jump in to Enter Protected Mode!!!

	I	Exam	ple v	vith Short Jump
;Causes	bx to		y 1 from bx	0 to 65535 to 0 to 65535 to
start:		bx, ax,	1 1	;Clear bx and initialize status flags ;ax < 1
peare.	add	ax,	bx	ax < ax+bx
	jmp	next		;add a displacement to IP ; (+2 from xor to mov)
	xor	bx,	bx	Clear bx and initialize flags;
	xor	ax,	ax	Clear ax and initialize flags;
next:	mov	bx,	ax	;bx < ax
	jmp	start		;add a displacement to IP
				; (a negative value - 2's comp.)









Conditional Control Instruction Summary Simple Flag Branches							
Jump b	ased o	n single	e fl	ag			
				CONDITIONAL			
jc	LABEL	;jump	on	carry (CF=1)			
jnc	LABEL	;jump	on	no carry (CF=0)			
je/jz	LABEL	;jump	if	ZF=1 - jump if equal/zero			
jne/jnz	LABEL	;jump	if	ZF=0 - jump not equal/jump if zero			
jo	LABEL	;jump	if	OF=1 - jump on overflow			
jno	LABEL	;jump	if	OF=0 - jump if no overflow			
js	LABEL	;jump	on	sign flag set (SF=1)			
jns		;jump	if	no sign flag (SF=0)			
jp/jpe		;jump	if	PF=1 - jump on parity/parity even			
jnp/jpo	LABEL	;jump	if	PF=0 - jump on no parity/parity odd			

Conditional Control Instruction Summary Branches for unsigned comparisons

Jump is based on flags used for unsigned number comparison (based on C, Z flag)

CONITIONAL ja/jnbcLABEL ; jump if CF=2F=0 - jump above-jump not below/equal jae/jnbcLABEL ; jump if CF=0 - jump above/equal-jump not below jb/jnacLABEL ; jump if CF=1 - jump below-jump not bove/equal jbe/jnaLABEL ; jump if CF=1 or ZF=1 - jump equal - jump zero

Typical use:

cmp al,bl jb there

; jump if al is 'below' bl ; unsigned comparison

Conditional Control Instruction Summary
Branches for signed comparisons
5
Jump is based on flags used for signed number
comparison (based on Z, S, V flags)
CONDITIONAL
jg/jnleLABEL ;jump if ZF=0 and (SF=OF) - jump greater/not less nor equal
jge/jnlLABEL ;jump if SF=OF - jump greater-equal/not less than
$jl/jngeLABEL$; jump if SF \neq OF - jump less than/not greater nor equal
jle/jngLABEL ;jump if ZF=1 or SF \neq OF - jump less or equal/not
great than jnc LABEL ;same as jae/jnb
Typical use:
cmp al,bl
jl there ; jump if al is less than bl
; signed comparison



Branch - Conditional Transfers

•Always SHORT jumps in 86-286

• Can be SHORT or NEAR in 386+

• Condition Tested is Content of SF, ZF, CF, PF, OF

CONDITIONAL SET 386+

•Result in Byte of Memory - Either 00h or 01h • Useful for Saving Flag Contents in Memory

EXAMPLE

 setb
 T1
 ;T1 <-- 01h if CF=1 else T1 <-- 00h</td>

 seto
 T1
 ;T1 <-- 01h if OF=1 else T1 <-- 00h</td>

 Lecrement cx and Conditional Jump

 • Combination of Decrement cx and Conditional Jump

 • Decrement cx and Conditional Jump

 • Decrement cx and f cx≠0 jumps to LABEL

 • 386+ loopw (ex operation) and loopd (exc operation)

 Example

 MDS
 PROC
 NEAR

 mov
 cit
 00

 mov
 cit
 00

 AGAIN:
 mov
 di

 add
 ax,
 (bx)

 mov
 di
 bx

 iods
 ax,
 (bx)

 iody
 aAGAIN:
 xet

 ADDS
 ENDP
 ENDP

Iteration Instruction, 100p						
ADDS	PROC mov mov cld mov lodsw add mov	NEAR cx, si, di, bx, ax, di,	<pre>100 ;cx < 64h - number of words to add OFFSET BLOCK1 ;si < offset of BLOCK1 (in ds) OFFSET BLOCK2 ;di < offset of BLOCK2 (in ds) ;Auto-increment si and di, DF=0 di ;bx < di, save offset of BLOCK2 ;ax < ds:[si], si<si+2, di<di+2<br="">[bx] ;ax < ax + ds:[bx] bx ;di < bx, restore di with ; offset in BLOCK2</si+2,></pre>			
ADDS	stosw loop ret ENDP	AGAIN	yes:[di] < ax, si <si+2, di<di+2<br="">;cx < cx - 1, if cx≠0 jump to AGAIN ;ip < ss:[sp]</si+2,>			

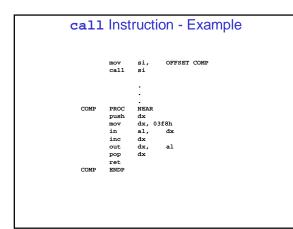
	Group of Instructions that Perform Single Task – (can be used as) a SUBROUTINE					
call ret	- invokes subroutine - pushes ip - returns from subroutine - pops ip					
• Uses MAS	M Directives: PROC and ENDP					
Must Spec	Must Specify					
NEAR FAR	- intrasegment - intersegment					
• Difference is op-code of ret						
NEAR FAR	- c3h - pops IP - cbh - pops CS, pops IP					

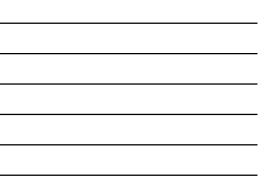
call Instruction

 \bullet Differs from jmp Since Return Address on Stack

- NEAR call:
 3 bytes 1 opcode and 2 for IP

 FAR call:
 5 bytes 1 opcode, 2 for IP and 2 for Cs
- call with operand can use 16-bit offset in any register except segment registers
 - call bx ;pushes ip then jumps to cs:[bx]





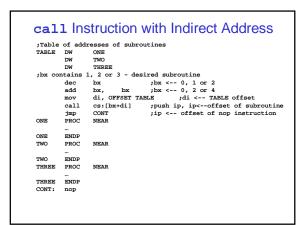
c	call Instruction - Example Explained							
	mov call	si, OFFSE si	I COMP ;get offset of COMP subroutine ;push ip, ip <si< td=""></si<>					
COMP	PROC push mov in	NEAR dx dx, 03f8h al, dx	;Save current contents of dx ;dx < 03f8h (an immediate data Xfer) ;al receives 1 byte of data from I/O					
	inc out	dx dx, al	<pre>; device with output port address 03f8h ;dx<03f9h ;send 1 byte of data to I/O device ; input port with address 03f9h</pre>					
COMP	pop ret ENDP	dx	;restore dx to value at call time ;ip <ss:[sp], sp<sp+2<="" td=""></ss:[sp],>					



call Instruction with Indirect Address

Useful for Choosing Different Subroutines at Runtime Can Use a Table (like the jump Table Example)								
<pre>;Assume bx contains 1, 2 or 3 for subroutine desired TABLE DW ONE DW TWO DW THREE dec bx add bx, bx mov di, OFFSET TABLE call cs:[bx+di] imp CONT</pre>								
ONE	PROC	NEAR						
one two two	 ENDP PROC ENDP	NEAR						
THREE THREE CONT:	PROC ENDP nop	NEAR						





ret	Inetri	CTION
TEL	nistiu	CUOIT

NEAR -	pops 16-bit value places in IP				
FAR -	pops 32-bit value places in CS:IP				

Type is Determined by PROC Directive
 Other Form of ret has Immediate Operand
 a) modifies SP before restoring IP
 b) Useful When Subroutine Modifies Stack

ΕΧΑΛ	sp 🔶 IP			
TEST	PROC push push	NEAR ax bx	;ss:[sp] <ax, sp<sp-2<br="">;ss:[sp]<ax, sp<sp-2<="" th=""><th></th></ax,></ax,>	
TEST	ret ENDP	 4	;ip <ss:[sp+4]< td=""><td>SP BX AX IP</td></ss:[sp+4]<>	SP BX AX IP