


16-Oct-06 (1)




CSC2510 - Computer Organization

Lecture 7: IA-32 and I/O

Philip Leong


16-Oct-06 (4)



Push/Pop

- PUSH src
 - $ESP \leftarrow ESP - 4$
 - $[ESP] \leftarrow src$
- POP dst
 - $dst \leftarrow [ESP]$
 - $ESP \leftarrow ESP + 4$
- PUSHAD
 - Pushes all 8 general purpose registers EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI on stack
- POPAD
 - Pops them (in reverse order) but doesn't modify ESP


16-Oct-06 (2)



Block Transfers

- REPINS & REPOUTS
 - Transfer block of items between memory and I/O
 - "S" suffix means string
 - "REP" prefix means repeat
 - Additional suffix "B" or "D" indicates size of transfer
 - DX is a 16-bit I/O device address
 - EDI is a 32-bit address for the start of the block in memory
 - ECX contains number of data items to be transferred
 - After each data item is transferred
 - EDI is incremented (by 1 or 4 depending on "B" or "D" suffix)
 - ECX decremented by 1
 - This is repeated until ECX=0

16-Oct-06 (5)




Parameter Passing via Registers

<pre> Calling program : : LEA EBX,NUM1 Load parameters : MOV ECX,N into EBX,ECX. : CALL LIST.ADD Branch to subroutine. : MOV SUM,EAX Store sum into memory. : Subroutine LIST.ADD: PUSH EDI Save EDI. : MOV EDI,0 Use EDI as index register. : MOV EAX,0 Use EAX as accumulator register. START.ADD: ADD EAX,[EBX + EDI * 4] Add next number. : INC EDI Increment index. : DEC ECX Decrement counter. : JG START.ADD Branch back if [ECX] > 0. : POP EDI Restore EDI. : RET Branch back to Calling program. </pre>	ESP →	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>[EDI]</td></tr> <tr><td>Return Address</td></tr> <tr><td>Old TOS</td></tr> </table>	[EDI]	Return Address	Old TOS
[EDI]					
Return Address					
Old TOS					

(a) Calling program and subroutine

16-Oct-06 (3)



Example


```

LEA EDI,MEMBLOCK
MOV DX,DISKDATA
MOV ECX,128
REPINS
                
```

Transfers 128 32-bit words from DISKDATA I/O register to [MEMBLOCK..MEMBLOCK+128*4-1]

What are the advantages of this instruction?

16-Oct-06 (6)



Parameter passing on Stack

(Assume top of stack is at level 1 below.)

<pre> Calling program : PUSH OFFSET NUM1 Push parameters onto the stack. : PUSH N Branch to the subroutine. : CALL LIST.ADD Remove n from the stack. : ADD ESP,4 Pop the sum into SUM. : POP SUM : Subroutine LIST.ADD: PUSH EDI Save EDI and use : MOV EDI,0 as index register. : PUSH EAX Save EAX and use as : MOV EAX,0 accumulator register. : PUSH EBX Save EBX and load : MOV EBX,[ESP+20] address NUM1. : PUSH ECX Save ECX and : MOV ECX,[ESP+20] load count n. START.ADD: ADD EAX,[EBX+EDI * 4] Add next number. : INC EDI Increment index. : DEC ECX Decrement counter. : JG START.ADD Branch back if not done. : MOV ESP,ESP+24 EAX Overwrite NUM1 in stack with sum. : POP ECX Restore registers. : POP EBX : POP EAX : POP EDI : RET </pre>	Level 3 →	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>[ECX]</td></tr> <tr><td>[EBX]</td></tr> <tr><td>[EAX]</td></tr> <tr><td>[EDI]</td></tr> <tr><td>Return Address</td></tr> <tr><td>n</td></tr> <tr><td>NUM1</td></tr> </table>	[ECX]	[EBX]	[EAX]	[EDI]	Return Address	n	NUM1
[ECX]									
[EBX]									
[EAX]									
[EDI]									
Return Address									
n									
NUM1									

(b) Stack contents at different times

(a) Calling program and subroutine

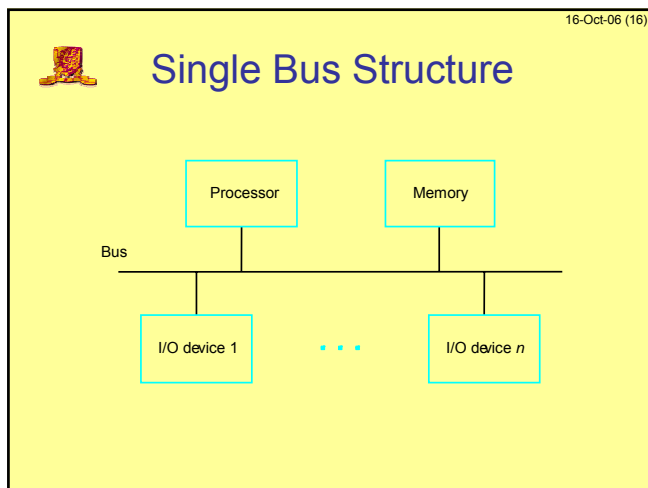
16-Oct-06 (13)

Linked list (insert)

Subroutine

INSERTION:	MOV RNEWID,[RNEWREC] CMP RHEAD,0 JG HEAD MOV RHEAD,RNEWREC RET	Check if list empty. If yes, new record becomes one-entry list. Check if new record becomes head.
HEAD:	CMP RNEWID,[RHEAD] JG SEARCH MOV [RNEWREC+4],RHEAD MOV RHEAD,RNEWREC RET	Check if new record becomes head. If yes, make new record the head.
SEARCH: LOOPSTART:	MOV RCURRENT,RHEAD MOV RNEXT,[RCURRENT+4] CMP RNEXT,0 JE TAIL CMP RNEWID,[RNEXT] JL INSERT MOV RCURRENT,RNEXT JMP LOOPSTART INSERT:	Otherwise, use RCURRENT and RNEXT to move through the list to find the insertion point.
TAIL:	MOV [RCURRENT+4],RNEXT MOV [RCURRENT+4],RNEWREC RET	

Figure 3.51. An IA-32 subroutine for inserting a new record into a linked list.



16-Oct-06 (14)

Linked List (delete)

Subroutine

DELETION:	CMP RIDNUM,[RHEAD] JGT SEARCH MOV RHEAD,[RHEAD+4] RET	Check if head. If yes, remove.
SEARCH: LOOPSTART:	MOV RCURRENT,RHEAD MOV RNEXT,[RCURRENT+4] CMP RIDNUM,[RNEXT] JEQ DELETE MOV RCURRENT,RNEXT JMP LOOPSTART DELETE:	Otherwise, use RCURRENT and RNEXT to move through the list to find the record.
	MOV RTEMP,[RNEXT+4] MOV [RCURRENT+4],RTEMP RET	

Figure 3.52. An IA-32 subroutine for deleting a record from a linked list.

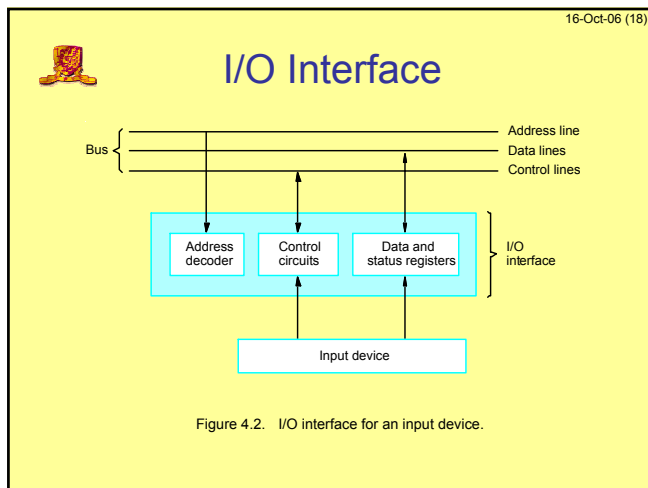
16-Oct-06 (17)

Addressing

- Memory mapped I/O – devices and memory share the same address space
- IA-32 can use memory mapped I/O and/or 16-bit I/O address space

16-Oct-06 (15)

Input/Output Organization



16-Oct-06 (19)

Example

Figure 4.3. Registers in keyboard and display interfaces.

16-Oct-06 (22)

Interrupt Example

- Overlap printing and computing
- When printer ready, sends an interrupt request
- This causes COMPUTE execution to be suspended and the interrupt routine PRINT to gain control
- PRINT interrupt routine executed and returns from the interrupt
- COMPUTE execution continued
- Has similarities with subroutine execution

Figure 4.5. Transfer of control through the use of interrupts.

16-Oct-06 (20)

Sample Program

Move	#LINE,R0	Initialize memory pointer.
TestBit	#0,STATUS	Test SIN.
Branch=0	WAITK	Wait for character to be entered.
Move	DATAIN,R1	Read character.
TestBit	#1,STATUS	Test SOUT.
Branch=0	WAITD	Wait for display to become ready.
Move	R1,DATAOUT	Send character to display.
Move	R1,(R0)+	Store character and advance pointer.
Compare	#SOD,R1	Check if Carriage Return.
Branch=0	WAITK	If not, get another character.
Move	#\$0A,DATAOUT	Otherwise, send Line Feed.
Call	PROCESS	Call a subroutine to process the input line.

- This is an example of program controlled-IO
- The processor **polls** the device

16-Oct-06 (23)

Interrupt Routines

- Interrupt-service routine
 - Interrupt request occurs during execution of instruction i
 - Processor first completes execution of i
 - Current PC pushed
 - Loads program counter with address of interrupt service routine (ISR) say at a fixed address
 - ISR runs and finally executes a return from interrupt instruction
 - PC popped to return to normal execution

16-Oct-06 (21)

Interrupts

- Polling does not allow for processor to do other things while it tests the device's status register
 - Could be doing other things
- Better to be interrupted when an event takes place e.g. when printer becomes ready

16-Oct-06 (24)

ISR

- The ISR must inform device that the interrupt request has been serviced
- Done through the device registers
 - Implicitly by say reading data that is available
 - Or explicitly by changing some status register
- ISR must also save and restore any registers that it uses or program that was interrupted would not work when we return to it
- The ISR must be short as possible to minimize **interrupt latency**

16-Oct-06 (25)



Interrupt Hardware

- Need to be able to handle an unknown number of possible interrupting devices
- $\text{INTR} = \text{INTR1} \text{ or } \text{INTR2} \text{ or } \dots \text{ INTR } n$

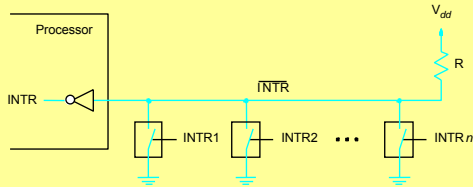


Figure 4.6. An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

16-Oct-06 (28)



Interrupt model

- Device raises interrupt request
- Processor interrupts program currently being accessed
- Interrupts disabled by clearing PS bit (except in the case of edge-triggered interrupts)
- Device is informed that its request has been recognised by ISR. In response it deactivates its interrupt request
- Action requested by interrupt performed in the ISR
- Interrupts enabled and execution of interrupted program resumed

16-Oct-06 (26)



Enable and Disable

- Sometimes need to ensure that interrupts cannot occur
 - May need to assign priorities e.g. high priority interrupts cannot be interrupted by low priority ones
 - Ensure that an active request doesn't lead to an infinite loop
 - Interrupt routines may need to access data structures and ensure they do not get interrupted while doing so

16-Oct-06 (29)



Multiple Devices

- What is multiple devices can interrupt processor?
 - How does the processor recognize which device is causing the interrupt?
 - How does it know which ISR to execute?
 - Can interrupts interrupt ISRs?
 - How do we handle simultaneous interrupt requests?
- Simple solution
 - Poll every possible interrupting device and check if status bit indicates something is to be done and then service it
 - Advantage: simple
 - Disadvantage: slow

16-Oct-06 (27)



Avoiding Infinite interrupt loops

- Can be handled by
 - Ignoring interrupts until after 1st instruction of ISR
 - Interrupt disable instruction is first instruction
 - No further interrupts can occur
 - Controlled by 1 bit in the processor-status (PS) register, when 1 will process interrupts, when 0 disables them
 - Set to 1 after the return from interrupt instruction
 - Processor automatically disable interrupts before starting ISR
 - Edge triggered interrupts. Processor receives one interrupt request per activation of the hardware interrupt line

16-Oct-06 (30)



Vectored Interrupts

- Interrupting device sends a code over the data bus to identify itself
- Processor jumps to a table of addresses, indexed by the interrupt-vector code

16-Oct-06 (31)

Interrupt nesting

- Interrupts usually assigned priorities
 - e.g. real-time clock (RTC) interrupt must respond before the next one whereas a keyboard interrupt has no such constraint
 - RTC should have higher priority
- During execution of an ISR, interrupts only accepted from higher priority devices
- Special **priority arbitration hardware** is used for this purpose

16-Oct-06 (34)

Example

- Suppose
 - Processor uses vectored interrupts
 - Starting address of ISR stored at INTVEC in memory
 - Interrupts enabled by setting bit 9 (the IE bit) in the processor status word
 - Keyboard and display devices are connected to the processor

Figure 4.3. Registers in keyboard and display interfaces.

16-Oct-06 (32)

Priority Schemes

- Priority is determined by order of polling
- Daisy chain, the INTA signal only passed on if the device does not have a request
 - Uses fewest wires but slower than ...
- Priority groups (most common in modern high speed processors)

16-Oct-06 (35)

Example

- To initialise the ISR
 - Load starting address of ISR to INTVEC
 - Load address LINE to PNTR (input chars will be stored here)
 - Enable keyboard interrupts (set bit 2 in register CONTROL)
 - Enable processor interrupts (set IE bit in processor status word PS)
- ISR
 - Read input character from keyboard input register (will cause the interface circuit to remove interrupt request)
 - Store character to location PNTR and increment PNTR
 - When end of the line reached, disable keyboard interrupts and inform program Main
 - Return from interrupt

16-Oct-06 (33)

Interrupt Device Requests

- A bit in the status flag that can be set/cleared in software enables/disables interrupts

16-Oct-06 (36)

Example

```

Main Program
-----
Move #LINE,PNTR      Initialize buffer pointer.
Clear EOL            Clear end-of-line indicator.
BitSet #2,CONTR_OL  Enable keyboard interrupts.
BitSet #9,PS         Set interrupt-enable bit in the PS.
:
:
In interrupt-service routine
-----
READ MoveMultiple R0-R1, -(SP) Save registers R0 and R1 on stack.
Move PNTR,R0        Load address pointer.
MoveByte DATAIN,R1 Get input character and
                    store it in memory.
MoveByte R1,(R0)+   Update pointer.
Move #SOD,R1        Check if Carriage Return.
CompareByte RTRN
Branch=0
Move #1,EOL         Indicate end of line.
BitClear #2,CONTR_OL Disable keyboard interrupts.
RTRN MoveMultiple (SP)+,R0-R1 Restore registers R0 and R1.
Return-from-interrupt
    
```