EECC
Instructor (240306)

Cándido Aramburu

2022-11-24

Table of Contents

I Arquitectura del Repertorio de Instrucciones (ISA): computadora von Neumann, datos, instrucciones, programación.

Chapter 1. Tareas

1.1. 2220

```
2018 Octubre 13
HOJAS DE REFERENCIA RAPIDA
  gdb
  ias
FAQ
GLOSARIO
ERRORES MAS COMUNES
  (ahora en el apéndice)
  llevar al FAQ
EXAMEN TIPO
EJERCICIOS
    Programación en C : punteros
    Algoritmia: pseudoinstrucción, organigramas
CAPITULOS:
 2-Arq von Neum: llevar tutoriales a ejercicios
 6-Prog ASM:Mnemónicos más utilizados
```

Chapter 2. Otros Apuntes

2.1. Teoría

- http://cs.indstate.edu/CS456/asmwikibook.pdf
- https://ocw.cs.pub.ro/courses/cns/labs/lab-02 → BUENIIIIISIMO
  - http://www.cs.virginia.edu/~evans/cs216/guides/x86.html → bueno
- http://opensecuritytraining.info/IntroX86.html
- https://en.wikibooks.org/wiki/X86_Assembly
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-172-performance-engineering-of-software-systems-fall-2010/video-lectures/lecture-6-c-to-assembler/
- http://www.cs.princeton.edu/courses/archive/spr08/cos217/schedule.html
- https://www.nand2tetris.org/
  - https://www.nand2tetris.org/course → pROJECTS
  - https://docs.wixstatic.com/ugd/44046b_bedcaf3edd954995b07527ab860f882b.pdf
  - LIBRO The Elements of Computing Systems Building a Modern Computer from First Principles
- https://chortle.ccsu.edu/AssemblyTutorial/index.html MIPS
  - https://en.wikipedia.org/wiki/SPIM

```
SPIM:

    SPIM is a MIPS processor simulator, designed to run assembly language
code for this architecture. The program simulates R2000.

    The name of the simulator is a reversal of the letters "MIPS".

    SPIM simulators are available for Windows (PCSpim), Mac OS X and
Unix/Linux-based (xspim) operating systems. As of release 8.0 in
January 2010, the simulator is licensed under the standard BSD
license.
```

2.2. Ejercicios

- http://www.fdi.ucm.es/profesor/mendias/512/512.html
  - /home/candido/Dropbox/apuntes/apuntes_Estr_Computadores/universidades/www.fdi.ucm.es/profesor/mendias/index-512.html

Chapter 3. Mecanismos de Entrada/Salida

3.1. Sincronizacion por Interrupcion

3.1.1. Proceso de atención a las interrupciones

Ejemplo: Procesamiento de la interrupción originada por el teclado.

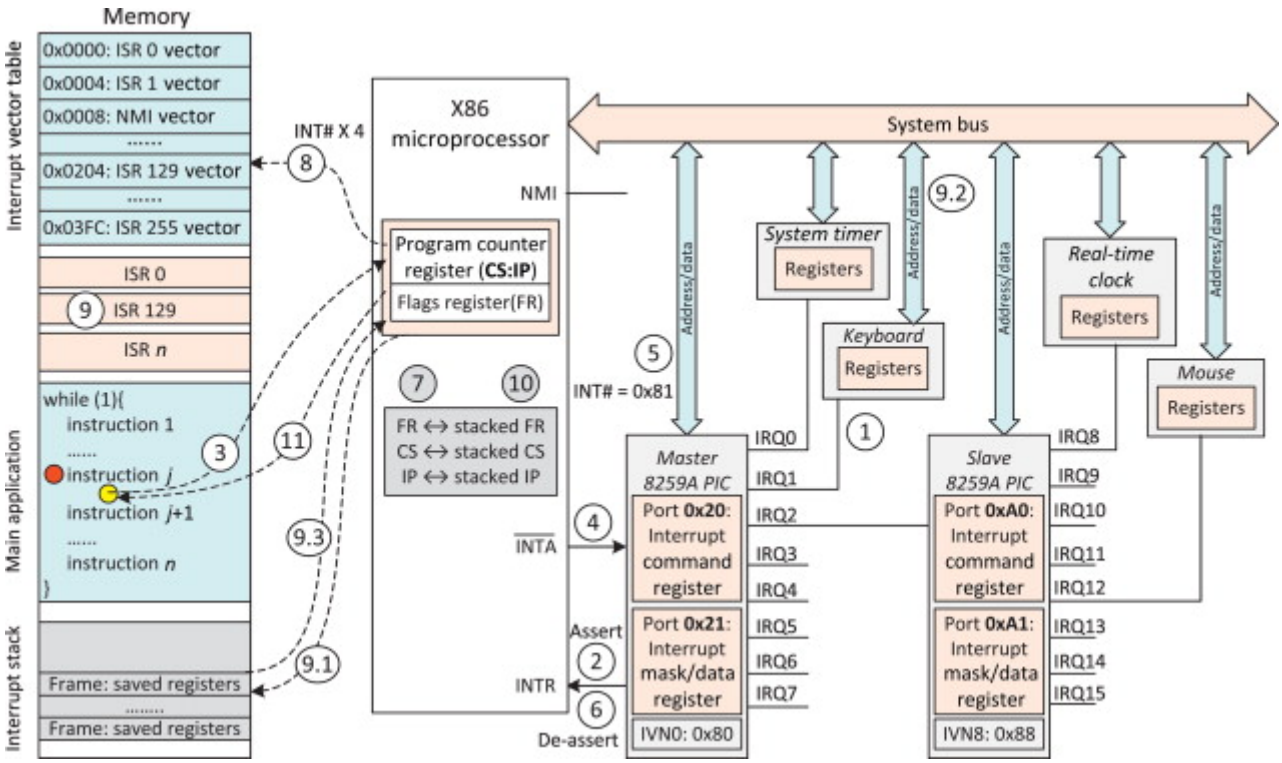

Figure 1. Interrupción del Teclado

1. Se pulsa una tecla y el controlador del teclado activa la señal IRQ1 del PIC
2. El PIC examina si la IRQ1 está autorizada a interrumpir y si no hay interrupciones de mayor prioridad. En tal caso, el PIC activa la entrada INTR de la CPU
3. La CPU interrumpe la ejecución del proceso main_application. Salva la dirección de retorno (PC) y desautoriza más interrupciones IRQ (Flag IF)
4. La CPU comprueba que el flag IF autoriza la interrupción, confirma al PIC la aceptación de la interrupción y solicita el vector de interrupción asociado a la IRQ1.
5. El PIC envía al bus de direcciones el vector 129 (0x81)
6. El PIC desactiva la solicitud de interrupción INTR a la CPU para poder gestionar otra IRQ.
7. La CPU guarda en el Interrupt Stack: (Flag Register,Code Segment,Instruction Pointer)
8. La CPU resetea el flag IF desautorizando ser interrumpido durante el cambio de contexto. Captura el vector de interrupción 129 de la tabla de vectores de interrupción→ captura de la tabla la entrada 129x4 que contiene la dirección de La Interruption Service Routine asociada al periférico teclado.
9. La CPU actualiza CS:IP para saltar a la dirección de la ISR 129 y comienza su ejecución
    a. La ISR salva los registros no modificables en el Interrupt Stack → Frame: Saved registers . Autoriza las interrupciones IF=1
    b. La ISR accede al periférico y se captura el valor de la tecla pulsada a través del data bus.
    c. Fin de la ISR . Se desautorizan las interrupciones durante la recuperación del anterior contexto. La CPU recupera el contexto del Interrupt Stack → Frame: Saved registers

10. Se recupera la dirección de retorno y se actualiza CS:IP
11. Continua la ejecución del programa interrumpido main_application
    - ScienceDirect.com → Interrupt Vector - an overview | ScienceDirect Topics
      - https://www.sciencedirect.com/topics/engineering/interrupt-vector

      ```
      The interrupt vectoring procedure of x86 processors is illustrated in Figure
      4.14. Shown in the top-left corner is the interrupt vector table, which
      starts at 0x0, the very beginning of the memory space, and ends at 0x03FF.
      The table has 256 interrupt vectors. Each interrupt vector has 4 bytes,
      containing 2 bytes for the IP register followed by 2 bytes for the CS
      register. The 4 bytes together, CS:IP, form an address, pointing to the
      location of an ISR.


      Each interrupt is associated with an interrupt vector number:

       For a hardware interrupt, its interrupt vector number is provided by the
      hardware device or PIC.
       For a software interrupt, its interrupt vector number is provided by the
      int instruction.
       For an internal interrupt, its interrupt vector number is fixed and is
      known by the processor.

      Given an interrupt vector number, the corresponding interrupt vector is
      located at the memory address corresponding to four times the interrupt
      vector number.

      The default x86 processor interrupt vector table is given on the left in
      Figure 4.15. The first 32 interrupt vectors (0x0-0x1F) are either reserved
      or defined by the x86 processor for internal interrupts (except 0x02, which
      is a vector for nonmaskable hardware interrupts). All the rest can be used
      for software interrupts and hardware interrupts.
      ```

    - The interrupting process shown in Figure 4.14 is explained below:

      ```
      1.
      A user presses a key on the keyboard, which drives an IRQ on the IRQ1 line of
      the 8259 master PIC.

      2.
      The PIC detects the IRQ on IRQ1. It changes the IRR by setting the bit
      corresponding to IRQ1. The PIC then examines the IMR to see if the interrupt
      source is disabled. If not, the PIC then determines if there is any higher-
      priority interrupt waiting to be serviced. If there is, this new IRQ has to wait
      until the higher-priority interrupt is serviced. If there is no higher-priority
      interrupt waiting to be serviced, the PIC stores the vector number 0x81 (the
      base 0x80 plus one offset) in an internal register, and asserts the INTR line to
      inform the processor.
      ```

```
3.
While the instruction j of the current program is being executed, the processor
samples INT and detects an asserted line. After the execution of the instruction
j has been completed, the current program is suspended. At this point, the value
contained by CS:IP is the location of the next instruction of the current
program.

4.
The processor examines the interrupt flag within the flags register. If the
interrupt flag is set, the processor acknowledges the IRQ by asserting the INTA
line to the PIC, expecting an interrupt vector number from the PIC.

5.
The PIC drives the interrupt vector number 0x81 to the system bus. It also sets
the corresponding bit inside the in-service register, indicating that interrupt
source 1 is currently being serviced.

6.
The PIC then deasserts the INTR line (so that a new IRQ can be asserted).

7.
The value of the flags register is pushed onto the interrupt stack. The values
of CS and IP are also pushed onto the interrupt stack, so that the original
program can be resumed later.

8.
To protect context switching, the “interrupt enable” flag of the flags register
is cleared to disable further interrupts. The processor comes to the keyboard
interrupt vector, which has 4 bytes, located at 4 x0x81 = 0x0204.

9.
The keyboard interrupt vector contains the address of the keyboard ISR. By
loading IP with the 16-bit data at 0x0204 and loading CS with the 16-bit data at
0x0206, the processor jumps to the start location of the keyboard ISR.

9.1.
Inside the ISR, the commonly used registers are first saved onto the interrupt
stack. At this point, the processor has switched its context from the last task
to the current ISR. The “interrupt enable” flag of the flags register is set to
enable further interrupts, if interrupt nesting is desired.

9.2.
The portion of code pertinent to the requesting device is executed. In this
case, the code of the key being pressed is stored in a memory area called the
keyboard buffer.

9.3.
At the end of the ISR, interrupts are again disabled for context switching. The
ISR first sends an EOI command to the master PIC’s command register. Upon
receiving this command, the PIC can clear the appropriate bit in the in-service
register, getting ready for new interrupts. The ISR then performs an instruction
```

```
to pop up the top frame of the interrupt stack.


10.
The context of the original task is restored. Especially, CS:IP now refers to
the next instruction of the original program.

11.
The processor is ready to resume the execution of the original program.
```

Chapter 4. Programación

4.1. Hola

- gnu cross compiler M68000

Chapter 5. Ejercicios

5.1. Hola

- x