Módulo 10
Programación en Lenguaje Ensamblador (Pt. 2)

Organización de Computadoras
Depto. Cs. e Ing. de la Comp.
Universidad Nacional del Sur

Copyright

Contenidos

- Estructura de un programa
- Las arquitecturas **i386** y **OCUNS**
- Estructuras de control
- Invocación de los servicios del **SO**
- Pasaje de parámetros
- Pila del programa y reentrancia
- Proceso de ensamblado, vinculación y carga

Llamadas al sistema

- El sistema operativo brinda sus servicios a los distintos programas bajo la forma de llamadas al sistema
  - Recordemos que el nivel 3 es híbrido, incorporando pocas instrucciones por sobre la funcionalidad provista por el nivel 2 de la máquina convencional
- La mayor parte de las llamadas al sistema involucran conceptos que serán abordados recién en otras asignaturas

Llamadas al sistema (CISC)

- La invocación de llamadas al sistema bajo el **GNU/Linux** es similar a la invocación de un procedimiento en **C**
  - En una arquitectura de 32 bits, la interrupción **80h** es el portal de acceso a todos los servicios de este **SO**
- El servicio solicitado se indica en el registro **EAX** y sus parámetros usualmente en los restantes registros del procesador
- Si el servicio retorna algún valor, lo hará también haciendo uso del registro **EAX**

Convenciones

- Por ejemplo, para invocar a la llamada al sistema **sys_write**, se requiere:
  - Cargar el número de servicio solicitado (**sys_write** es el **4**) en el registro **EAX**
  - Cargar el descriptor del archivo que queremos escribir en el registro **EBX**
  - Cargar el puntero al buffer conteniendo lo que queremos escribir en el registro **ECX**
  - Cargar la cantidad de bytes que queremos escribir en el registro **EDX**

Convenciones

- Por lo general el pasaje de parámetros en las llamadas al sistema presenta dos variantes, en función de la cantidad de argumentos:
  - Para pasar de cero a seis argumentos se usa pasaje de parámetros en registros, usando los registros **EBX**, **ECX**, **EDX**, **ESI**, **EDI** y **EBP**, en ese orden
  - Para pasar más de seis argumentos se usa pasaje de parámetros en memoria, usando el registro **EBX** para indicar la dirección de comienzo de la lista de parámetros

Hola mundo! (CISC)

```
section .data
texto db 'Hola mundo!',10
largo equ $ - texto ; $ denota el offset
                    ; actual

section .text
global _start ; etiqueta global que
              ; marca el comienzo
              ; del programa
```

Hola mundo! (CISC)

```
_start:
mov eax, 4      ; servicio sys_write
mov ebx, 1      ; standard output
mov ecx, texto  ; offset de mensaje
mov edx, largo  ; largo del mensaje
int 80h         ; invocación al servicio
mov eax, 1      ; servicio sys_exit
mov ebx, 0      ; terminación sin errores
int 80h         ; invocación al servicio
```

Llamadas al sistema (RISC)

- En el caso de la arquitectura **OCUNS**, el tercer nivel del sistema operativo extiende mínimamente a la máquina convencional
- Solo se asume que los accesos a la posición de memoria **FF** serán capturados y redireccionados al dispositivo estándar de entrada o de salida según corresponda
  - Por caso, lo escrito en esa posición saldrá por pantalla y lo leído será lo ingresado por el teclado

Hola mundo! (RISC)

- Tarea para el hogar: desarrollar un programa para la arquitectura **OCUNS** que sea equivalente al presentado para la arquitectura **i386**
  - El principal desafío consiste en resolver cómo generar la secuencia de caracteres a ser escritos en la posición de memoria **FF**

Invocación a rutinas (CISC)

- La invocación a un procedimiento no difiere en gran medida de los saltos convencionales salvo en un detalle puntual
- Se adopta la convención de dejar en el tope la pila del programa la dirección a la que se debe retornar una vez finalizada la ejecución del procedimiento
  - Las instrucciones **`call`** y **`ret`** simplifican la invocación a procedimientos y el posterior retorno una vez finalizados

Ejemplo

```
start_:
    mov eax, [UNVALOR]
    call triplicar      ; procedimiento para triplicar
    ...                 ; el valor almacenado en EAX
triplicar:
    push ecx            ; preservo ECX en la pila
    push edx            ; preservo EDX en la pila
    mov ecx, 3
    mul ecx             ; multiplico por 3
    pop edx             ; recupero EDX de la pila
    pop ecx             ; recupero ECX de la pila
    ret                 ; retorno
```

Invocación a rutinas (RISC)

- La arquitectura **OCUNS** también cuenta con un par de instrucciones a tal efecto
- Cabe acotar que es diferente el mecanismo para comunicar cuál es la dirección de retorno, ya que se hace uso de un registro en vez de usar la pila del programa
  - En caso de implementar programas reentrantes, la pila puede ser simulada por el programador
  - La ley de equivalencia entre hardware y software está siempre al alcance de la mano del programador

Ejemplo

```
        lda RE, FF
        load RA, 0(RE) ; lee un valor en RA
        call dupli, R0 ; invoca la rutina
        store RA, 0(RE); se muestra en pant.
        hlt

dupli:  add RA, RA, RA ; ¿qué parámetros
        jmp R0         ; recibe dupli?
```

Pila del programa

- La pila del programa (stack) es una de las regiones que forma parte de los programas en ejecución
  - Su tamaño es dinámico, crece o disminuye según se requiera
  - Crece hacia abajo desde la dirección más alta
  - Es utilizada para almacenar datos temporalmente, tales como la dirección de retorno al invocar a un procedimiento o el valor actual de un registro el cual será alterado, por caso, durante la invocación a un servicio del sistema operativo

Pila del programa (CISC)

- Operaciones sobre la pila del programa provistas por la arquitectura **i386**:
  - Instrucción **push**: resta el tamaño en bytes del objeto apilado al valor actual del registro **ESP** (stack pointer) y lo almacena en el lugar ahora apuntado por **ESP**
  - Instrucción **pop**: recupera el objeto a ser desapilado de la locación apuntada por el registro **ESP** y lo actualiza sumándole el tamaño en bytes de ese objeto. Nótese que el objeto sigue en memoria, pero será sobrescrito cuando el tamaño de la pila vuelva a crecer (es decir, cuando **ESP** disminuya)

Pila del programa (RISC)

- En la arquitectura **OCUNS** la pila del programa debe ser gestionada por el programador
- La idea es reservar un registro y hacer uso del mismo como si se tratara de un puntero al tope de la pila
  - El registro sigue siendo de propósito general, es decir, la arquitectura no impide que se adultere el valor del puntero de manera deliberada

Caller save vs. callee save

- Para preservar el contenido de los registros que serán alterados durante la ejecución de un procedimiento surgen dos alternativas:
  - Una posibilidad (denominada “caller save”) es que quien llama al procedimiento se encargue de salvar en la pila del programa aquellos registros que se necesiten preservar
  - Otra posibilidad (denominada “callee save”) es que el procedimiento que es llamado se encargue de preservar en la pila del programa sólo aquellos registros que serán modificados

Caller save vs. callee save

- Cada alternativa tiene ventajas y desventajas:
  - Caller save permite optimizar múltiples llamados a un procedimiento, evitando que cada una de las invocaciones preserve y recupere de la pila múltiples registros
  - Callee save, por otra parte, posibilita que el procedimiento sólo preserve los registros que serán afectados (el llamador usualmente no tiene acceso a esta información)
- La idea es adoptar sólo uno de estos esquemas a lo largo de un mismo programa

Pasaje de parámetros

- Los procedimiento en ocasiones requieren recibir información adicional al ser invocados, esto es, requieren un conjunto de parámetros
- El pasaje de estos parámetros se puede resolver de diversas formas:
  - Usando los registros del procesador
  - Usando una región de la memoria
  - Usando la pila del programa

Pasaje de parámetros

- Usando registros:
  - Este es por lejos el mecanismo más eficiente (pues no implica accesos adicionales a memoria)
  - La cantidad de parámetros está limitada a la cantidad de registros disponibles
- Usando la memoria:
  - Este mecanismo es más flexible que el anterior, pues permite una cantidad mucho mayor de parámetros
  - Naturalmente, tiene como desventaja que implica accesos adicionales a memoria

Pasaje de parámetros

- Usando la pila del programa:
  - El pasaje de parámetros usando registros o usando la memoria no resulta del todo apropiado a la hora de implementar procedimientos recursivos
  - En contraste, el pasaje de parámetros usando la pila del programa permite la implementación de procedimientos recursivos con relativa simpleza
  - Nótese que como la dirección de retorno suele ocupar el tope de la pila, los parámetros serán apilados por debajo (¡lo que implica que la pila no será accedida como una pila!)

Reentrancia

- ¿Por qué resulta más conveniente usar la pila, que está memoria, que directamente un región de la memoria?
- En otras palabras, ¿por qué en un caso podemos implementar soluciones recursivas y en el otro no?
- La diferencia entre las dos alternativas radica en que usando la pila obtenemos lo que se denomina un procedimiento reentrante o puro

Reentrancia

- Para alcanzar la reentrancia hace falta que el código no sea automodificable y que sólo se opere sobre los registros o la pila del programa
  - Para apreciar las ventajas de este tipo de código hay que recordar que los sistemas en la actualidad son multiprogramados, es decir, existe más de un programa en ejecución a la vez
  - Bajo esta definición, la recursión consiste meramente en un mismo programa sacando provecho de la reentrancia de una determinada rutina

Ejecución de un programa

- Como hemos visto, un programa atraviesa distintas etapas desde su concepción hasta llegar a estar corriendo en una arquitectura:
  - En primer lugar hay que escribir el código fuente
  - Más tarde hay que convertir ese código fuente en código objeto
  - Luego, se debe convertir el código objeto (de uno o más archivos fuente) en el código ejecutable
  - Finalmente, se debe cargar en memoria el código ejecutable y comenzar su ejecución

Etapa de compilación

- Características:
  - Durante esta etapa el código fuente es convertido en código objeto instrucción por instrucción
  - Se resuelven la totalidad de las referencias locales
  - Se identifican y dejan pendientes de resolver a la totalidad de las referencias externas
  - El código objeto es prácticamente ejecutable, sólo resta incorporar el código objeto asociado a otros módulos o bien a las funciones de librería que se hayan utilizado

Etapa de vinculación

- Características:
  - Durante esta etapa el código objeto de uno o más archivos fuentes es convertido en código ejecutable
  - La principal tarea del vinculador es resolver la totalidad de las referencias externas
  - Las referencias a otros módulos se resuelven a partir de código objeto de ese módulo (el cual debe suministrarse en ese momento)
  - Las referencias a librerías se resuelven usando el código objeto de la librería, el cual estará disponible en alguna ubicación conocida de antemano

Etapa de carga

- Características:
  - El cargador, un componente del sistema operativo, es el encargado de cargar a memoria el código ejecutable de un cierto programa
  - A esta altura todas las referencias externas están resueltas, sólo resta conocer la dirección inicial a partir de la cual será cargado el código ejecutable
  - Los modelos más avanzados de manejo de memoria han simplificado notablemente esta tarea (por caso, al contar con un espacio de direccionamiento de uso exclusivo para cada proceso)

Vinculación dinámica

- Como es frecuente que múltiples programas compartan la misma librería (por caso, **`stdio`**), en la actualidad es posible vincular múltiples programas a una misma instancia en memoria de la librería
  - Desde ya, esta librería debe estar compuesta exclusivamente de procedimientos reentrantes
  - Nótese que esas referencias externas no podrán ser resueltas en tiempo de vinculación, razón por la cual es este tipo de librería se las denomina de vinculación dinámica (por caso, las librerías **`.DLL`**)

¿Preguntas?