Segundo parcial de Arquitectura de Computadoras

22 de noviembre de 2022

Instrucciones:

- **Indique su nombre, apellido y número de cédula en todas las hojas que entregue.**
- **Escriba las hojas de un solo lado. Empiece cada ejercicio en una hoja nueva.**
- **Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.**
- **Apague su celular. No puede utilizar material ni calculadora.**
- **La duración del parcial es de una hora y media. En dicho tiempo debe también completar sus datos.**

Pregunta 1

Se sabe que la dirección de memoria de una caché, con función de correspondencia asociativa por conjuntos de 4 vías, se interpreta de la siguiente manera:

Dirección: TAG (16 bits) | SET (10 bits) | BYTE (10 bits).

Indique el tamaño y organización de la caché. Justifique su respuesta.

Pregunta 2

En el contexto de una CPU con pipeline, explique qué es un hazard de control. Describa dos posibles técnicas para mitigarlos.

Pregunta 3

Dada la siguiente estructura de datos en una arquitectura x86:

```
typedef struct {
     char dia;
     char mes;
     short anio;
     char nombre[32];
} agenda;

agenda cumpleanios[10];
```

a) Detalle cómo se organiza en memoria la variable *cumpleanios* sabiendo que se ubica a partir de la dirección 0x100.

b) Indique la dirección de memoria de cumpleanios[5].mes.

Pregunta 4

Describa el mecanismo de atención a interrupciones en un computador 8086 que dispone de controlador de interrupciones.

Pregunta 5

a) Indique el valor de CX luego del pop en el procedimiento principal
b) Detalle cómo se va modificando el stack, incluyendo las direcciones de memoria y los contenidos de cada elemento, hasta el lugar indicado en el código. El valor inicial de SP, al momento de comenzar a ejecutar “principal”, es 0xF03A.

```
proc principal
   mov AX, 0x1514
   mov BX, 0x0AEC
   push AX
   push BX
   call func
   pop cx
endproc


proc func
   push BP
   mov BP, SP
   mov AX, [BP + 4]
   mov BX, [BP + 6]
   add AX, BX
         ; detallar el estado del stack hasta aquí
   mov [BP+6], AX
   mov AX, [BP+2]
   mov [BP+4], AX
   pop BP
   add SP, 2
   ret
endproc
```

Solución

Respuesta Pregunta 1

La cantidad de líneas de una caché con función de correspondencia asociativa por conjuntos son la cantidad de conjuntos multiplicado por la cantidad de líneas por conjunto.

Dado que el campo conjunto (set) es de 10 bits, es posible identificar todos los conjuntos de la caché con 10 bits, por lo que se tienen 2^{10} conjuntos posibles. Dado que los conjuntos son de 4 vías, cada conjunto tiene 4 líneas, es decir 2^2 líneas. Por ende:

#conjuntos * #lineas_por_conjunto = #líneas_cache => $2^{10} . 2^2 = 2^{12}$ líneas

Luego, el tamaño de la cache es la cantidad de lineas multiplicado por la cantidad de bytes por linea.

Dado que el campo byte es de 10 bits, se tienen 2^{10} bytes por linea. Por ende:

#líneas_cache * #bytes_por_ línea = #tamaño_cache => $2^{12} . 2^{10} = 2^{22}$ bytes

Tamaño de la cache: 4 MB

Organización de la cache: 2^{12} líneas (4096 líneas) de 2^{10} bytes (1024 bytes)

Respuesta Pregunta 2

Un hazard es un problema o situación que genera problemas para el funcionamiento óptimo del pipeline. Los hazards de control son aquellos causados por instrucciones de salto, ya sean condicionales o incondicionales, u otras modificaciones del registro IP. La etapa de Fetch asume, por defecto, que la próxima instrucción a ejecutarse es la que está a continuación en memoria, lo que nunca es correcto si la instrucción es un salto incondicional ó no es siempre correcto si se trata de un salto condicional. Esto genera que se deba descartar la instrucción leída en la operación de fetch completada e iniciar un nuevo fetch a la nueva dirección (la determinada por el salto).

Dos posibles técnicas de mitigación son:

- Prefetch del destino:

Esta técnica consiste en realizar simultáneamente el fetch de la próxima instrucción en memoria y de la instrucción apuntada por la dirección destino del salto. Cuando el salto se efectiviza la instrucción que no corresponde a la secuencia lógica se descarta.

- Múltiples flujos:

Este caso va un poco más allá del anterior y duplica las etapas del pipeline que están ubicadas hasta la que decide el salto. De esta manera se puede avanzar en la ejecución de ambos flujos de ins-

trucciones, por lo que al momento de tomar el salto no hay ninguna penalización, ya que no es necesario “vaciar” el pipeline (en realidad se vacía uno de los dos: el que estaba ejecutando el flujo que finalmente no resultó elegido por el salto).

Respuesta Pregunta 3

a)

0x100	cumple[0].dia
0x101	cumple[0].mes
0x102	cumple[0].anio.parte_baja
0x103	cumple[0].anio.parte_alta
0x104	cumple[0].nombre[0]
0x105	cumple[0].nombre[1]
0x106	cumple[0].nombre[2]
0x107	cumple[0].nombre[3]
	....
0x123	cumple[0].nombre[31]
0x124	cumple[1].dia
0x125	cumple[1].mes
0x126	cumple[1].anio.parte_baja
0x127	cumple[1].anio.parte_alta
0x128	cumple[1].nombre[0]
0x129	cumple[1].nombre[1]
0x12A	cumple[1].nombre[2]
0x12B	cumple[1].nombre[3]
	....
0x147	cumple[1].nombre[31]
0x148	cumple[2].dia
0x149	....

b)

Cada elemento de agenda ocupa 36 bytes (0x24), por ende:

cumple[0] → dirección 0x100

cumple[1] → dirección 0x124

cumple[2] → dirección 0x148

cumple[3] → dirección 0x16C

cumple[4] → dirección 0x190

cumple[5]→ dirección 0x1B4

cumple[5].dia→ dirección 0x1B4

cumple[5].mes → dirección 0x1B5

Respuesta Pregunta 4

Para atender una solicitud de interrupción, la CPU realiza los siguientes pasos:

- El controlador de E/S que desea ser atendido envía la solicitud al controlador de interrupciones, que recibe el pedido a través de una entrada IRQ de éste. El controlador de interrupciones entonces genera un pedido a la CPU a través de la señal INT.

- La CPU "termina" de ejecutar la instrucción actual y si las interrupciones están habilitadas chequea si hay algún pedido. Notemos que utilizamos las comillas para señalar que, en realidad, lo que ocurre es que la CPU consulta si hay un pedido de interrupción al final del ciclo de instrucción, luego de la etapa de "write" y antes del siguiente "fetch". Esto se traduce en que el pedido de interrupción debe esperar hasta el final de la ejecución de la instrucción para que sea efectivamente detectado como tal.

- Salva el registro de las flags, el registro IP y el registro CS, a través del stack. CS e IP se guardan como forma de poder regresar a ejecutar la siguiente instrucción del programa que fue interrumpido, luego de la ejecución de la rutina de servicio a la interrupción, y como el modelo de memoria es segmentado es necesario guardar ambos valores (puntero FAR).

- Identifica el controlador de E/S que realizó el pedido de interrupciónactivando la señal INTA para que el controlador de interrupciones coloque en el bus de datos la identificación asociada a la entrada IRQ por la que llegó el pedido.

- Obtiene la dirección de la rutina de servicio de la interrupción correspondiente al controlador identificado. Se usa el identificador del pedido de interrupción como índice a una tabla (el vector de interrupciones) que en cada entrada tiene la dirección absoluta de la rutina de interrupción asociada, formada por el segmento (valor a cargar en el registro de segmento de código) y el desplazamiento dentro del segmento (valor a cargar en el puntero de instrucción) de la dirección de memoria de la primer instrucción de la rutina de interrupción.

- Enmascara las interrupciones. Esto significa que inhibe la aceptación de nuevos pedidos de interrupción hasta tanto ocurran uno de los siguientes eventos: se retorne de la rutina de servicio de la interrupción (todas las arquitecturas tienen una instrucción específica para hacer esto) ó el código de la rutina de servicio habilite en forma explícita (a través de una instrucción) la posibilidad de aceptar nuevas interrupciones.

- Pasa a ejecutar la rutina de servicio a la interrupción, cargando en CS:IP (puntero de instrucción) la dirección obtenida del vector de interrupciones, que es la dirección de la primer instrucción de dicha rutina.

Respuesta Pregunta 5

a) 0x2000 (que es la suma de 0x1514 y 0x0AEC que es la operación que realiza la función)

b)

	push AX	push BX	call func	push BP
SS:0xF03A				
SS:0xF038	0x1514	0x1514	0x1514	0x1514
SS:0xF036		0x0AEC	0x0AEC	0x0AEC
SS:0xF034			IP	IP
SS:0xF032				BP
SS:0xF030				