Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Ordenadores

2º I. T. Telecomunicación

Examen extraordinario - 10 de julio de 2.001
Soluciones

Problema 1

Apartado 1

Codificando la microinstrucción por campos, tenemos que cada palabra de microinstrucción estará formada por tres partes: los campos que representan las señales a activar; la condición de salto; y la palabra de microinstrucción siguiente a ejecutar.

Las señales de control a activar pueden agruparse en los campos que se muestran en la tabla 1. En esa tabla se supone que el número `n` de registros de propósito general es de 4; en el caso general, el campo que codifica las señales de control de entrada al bus tendrá $log_2(n+8)$ bits, redondeado hacia arriba. En el caso de las señales que gobiernan la salida de datos al bus, en el caso general harán falta $log_2(n+6)$ bits para codificarlas, redondeado también hacia arriba. Respecto de las señales "Operación", en el enunciado se indica que se codifican a través de tres señales. No se incluye aquí la señal NoOp porque puede regularse la salida de la ALU a través de la señal Yout. La señal Yin se ha considerado de salida por simplificar, aunque podría codificarse separadamente. Supondremos también que la señal de Flagsin se activa conjuntamente con Yin.

Apartado 2

La fase de búsqueda de cada una de las tres instrucciones es muy similar, ya que las tres constan de un primer byte con el código de operación y un segundo byte con el operando. La única diferencia estriba en que en el caso del `ADD` el direccionamiento es indirecto. El fragmento del microprograma que activa las señales de control necesarias para ejecutar las instrucciones es el siguiente:

`LD 0x5`

1. Fase de búsqueda:
   (a) PCout, MARin
   (b) R, WMFC
   (c) MBout, IRin, PC++
   (d) PCout, MARin
   (e) R, WMFC, PC++
2. Fase de ejecución:
   (a) MBout, Xin

`ADD 0x12`

1. Fase de búsqueda:
   (a) PCout, MARin
   (b) R, WMFC
   (c) MBout, IRin, PC++
   (d) PCout, MARin
   (e) R, WMFC, PC++
   (f) MBout, MARin
   (g) R, WMFC
2. Fase de ejecución:
   (a) MBout, ALUin1, ALUin2
   (b) Operación de suma, Yin
   (c) Yout, Xin

`ST 0x100`

Debido a una errata, aquí el operando tiene 9 bits, por lo que no podría codificarse en 8. Suponiendo que sí se pudiera, la secuencia de señales de control sería la siguiente:

1. Fase de búsqueda:
   (a) PCout, MARin
   (b) R, WMFC
   (c) MBout, IRin, PC++
   (d) PCout, MARin
   (e) R, WMFC, PC++
2. Fase de ejecución:
   (a) MBout, MARin
   (b) Yout, MBin
   (c) W, WMFC

Apartado 3

Si utilizamos la codificación propuesta en la tabla 1, la codificación de las fases de ejecución de las instrucciones anteriores nos dan como resultado los siguientes vectores de señales de control. Dado que no se conoce la codificación utilizada para la operación de suma de la ALU, dicha codificación se deja indicada como "???".

Fase de ejecución de `LD 0x05`						
1010	0011	00	000	0	0	0

Fase de ejecución de `ADD 0x12`						
1011	0011	00	000	0	0	1
0000	1001	00	???	0	0	0
1010	1010	00	000	0	0	0

Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Entrada del bus	Salida al bus	Incr/Decr	Operación	R	W	ALUin1
0000 NoOp	0000 NoOp	00 NoOp	000	0 NoOp	0 NoOp	0 NoOp
0001 IRin	0001 PCout	01 PC++	001	1 R	1 W	1 ALUin1
0010 PCin	0010 SPout	10 SP++	010			
0011 SPin	0011 MBout	10 SP--	011			
0100 MARin	0100 R1out		100			
0101 MBin	0101 R2out		101			
0110 R1in	0110 R3out		110			
0111 R2in	0111 R4out		111			
1000 R3in	1000 Xout					
1001 R4in	1001 Yin					
1010 Xin	1010 Yout					
1011 ALUin2						
4 bits	4 bits	2 bits	3 bits	1 bit	1 bit	1 bit

Tabla 1: Codificación por campos de las señales de control del problema 1.

Fase de ejecución de `ST 0x100`						
0100	0011	00	000	0	0	0
0101	1010	00	000	0	0	0
0000	0000	00	000	0	1	0

Problema 2

Apartado 1

Aplicando el algoritmo de Huffman, obtenemos una codificación como la siguiente. (Puede haber diferencias en el código obtenido, aunque no en el número de bits que componen el código de cada instrucción.)

40%	0	40	0	40	0	40	0	40	0
27%	10	27	10	27	10	27	10	60	1
13%	110	13	110	13	110	33	11		
10%	1110	10	1110	20	111				
5%	11110	10	1111						
5%	11111								

Apartado 2

La técnica de los códigos de operación con extensión se basa en asignar códigos cortos a las instrucciones más probables, y reservar combinaciones para extender el código. Por lo tanto, dado que se trata de un método empírico, hay varias soluciones posibles.

En nuestro caso, si asignamos dos bits a las instrucciones más frecuentes y reservamos una de las cuatro combinaciones posibles para extender el código, obtenemos:

40%	00
27%	01
13%	10
10%	110
5%	1110
5%	1111

Si en lugar de reservar una combinación reservamos dos, obtenemos lo siguiente:

40%	00
27%	01
13%	100
10%	101
5%	110
5%	111

Finalmente, si asignáramos tres bits, nos queda lo siguiente:

40%	000
27%	001
13%	010
10%	011
5%	100
5%	101

Como puede verse, con tres bits podríamos codificar todas las instrucciones, lo que no se ajusta al principio de que "las instrucciones más probables deberán tener códigos de operación más cortos". Es preferible asignarle dos bits a la instrucción más probable.

Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Apartado 3

El método de Huffman permite automatizar la asignación de códigos de operación más cortos a las instrucciones más frecuentes. Sin embargo, genera códigos demasiado largos para las instrucciones menos frecuentes. El método de los códigos de operación con extensión, por su parte, se basa en una asignación empírica de los códigos, pero se aprovecha mejor el espacio de codificación al permitir una asignación arbitraria de las combinaciones.

Apartado 4

Sí: es lógico que las probabilidades dinámicas de ejecución sean similares, ya que, dado un intervalo de tiempo suficientemente grande, a cada llamada a subrutina le debe corresponder un retorno de subrutina. Sin embargo, esto no tiene por qué ocurrir si las probabilidades son estáticas, es decir, si los números indican la probabilidad de aparición de cada instrucción en el código de un programa. Si las probabilidades medidas fueran estáticas, la probabilidad de encontrar un `call` sería mayor o igual que la de encontrar un `ret`, ya que por cada subrutina habrá una o más llamadas a la misma (a menos que haya subrutinas que no se utilicen, lo cual no tendría sentido).

Cuestión 1

Directo de registro: Se codifica el registro donde está el operando. Por lo tanto, el operando está en el registro indicado.

Directo: El operando es una posición de memoria. El direccionamiento indica dicha posición.

Indirecto: El operando indica la localización de la dirección efectiva del operando. Tipos: indirecto por registros (un registro que almacena la dirección de memoria en donde está el operando), indirecto de memoria (una dirección de memoria que almacena la dirección de memoria en donde está el operando).

Relativo: Se toma como referencia un punto de la memoria, y el operando se indica a través de un desplazamiento sobre dicho punto. Típicamente, el direccionamiento es relativo a un registro, al que se le suma un *offset*.

Por base y desplazamiento: Tipo de direccionamiento indirecto en el que se utiliza un registro base (que está implícito) y el offset se indica explícitamente. La dirección efectiva se calcula sumando ambos.

Cuestión 2

Los bits de suma y acarreo de un sumador total vienen dado por las siguientes ecuaciones:

$$s_i = (a_i \oplus b_i) \oplus c_i$$
$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

Definimos las variables

$$G_i = a_i b_i \text{ (Variable generada)}$$
$$P_i = a_i \oplus b_i \text{ (Variable propagable)}$$

En función de estas dos variables, las dos ecuaciones para s_i y c_{i+1} pueden escribirse como

$$s_i = P_i \oplus c_i$$
$$c_{i+1} = a_i b_i + (a_i + b_i) c_i = G_i + P_i c_i$$

La segunda ecuación vista, $G_i + P_i c_i$, se basa en suponer que P_i es equivalente a $(a_i + b_i)$. Esto es cierto para las combinaciones (0 0), (0 1) y (1 0). En el caso de que $a_i = b_i = 1$, se da la circunstancia de que G_i también valdrá 1, por lo que el resultado de la ecuación vista será 1, independientemente del valor del segundo término, $P_i c_i$.

Un sumador con acarreo anticipado realiza la suma en tres pasos:

1. Las células sumadoras calculan, todas a la vez, las variables P_i y G_i.
2. Con todos los valores de P_i y G_i ya disponibles, se calculan todos los acarreos.
3. Con los acarreos y los valores de a_i y b_i se calculan simultáneamente todos los bits de suma.

Cuestión 3

El uso de un "bus único" o "bus del sistema" consiste en utilizar el mismo bus para acceder a los dispositivos y a la memoria. Con esta organización pueden usarse dos técnicas de selección de dispositivo:

E/S aislada: Existe una línea auxiliar de control (!IO/M), que indica si se trata de una operación de acceso a memoria o de acceso a un periférico de E/S. Si el acceso es a un dispositivo, se utilizan las líneas de orden más bajo del bus de direcciones para seleccionar el dispositivo. Requiere usar instrucciones específicas para acceder a los dispositivos.

E/S mapeada: Esta técnica utiliza parte del espacio direccionable para acceder a la memoria y el resto para acceder a periféricos. Por lo tanto, los periféricos ocupan una parte del espacio direccionable, contrariamente a la E/S aislada. Pueden usarse las mismas instrucciones que para acceder a memoria.

Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Ventajas e inconvenientes: la E/S aislada facilita la identificación de las instrucciones de E/S en un programa, y permite distinguir claramente entre accesos a memoria y a la E/S: los puertos de E/S son normalmente mucho más lentos, unidireccionales y muchas veces incapaces de retener el contenido enviado.

Por otra parte, la E/S por mapa permite al programador utilizar el juego de instrucciones de acceso a memoria (es mucho más ortogonal), facilitando el flujo de información entre la memoria y la E/S. Además, el espacio de E/S es mas fácil de manejar al requerir menos señales de control.

Cuestión 4

La estructura de interrupciones del 8086 permite hasta 256 vectores de interrupción diferentes. Se dispone de una tabla de 256 entradas que comienza a partir de la dirección `0:0`, con entradas de 4 bytes. Esos cuatro bytes representan el segmento en donde se encuentra la rutina de atención y el desplazamiento dentro de ese segmento.

Los pasos necesarios para la atención de interrupciones vectorizadas son los siguientes:

1. El dispositivo solicita la interrupción a través de la línea INTR.
2. El procesador termina la ejecución de la instrucción en curso y analiza el estado de INTR (si no estuviera actualizada, continuaría normalmente).
3. La CPU reconoce la interrupción, activando INTA.
4. El dispositivo que recibe la INTA envía el código de interrupción por el bus de datos.
5. La CPU calcula a partir de este código la dirección de memoria en donde se encuentra la rutina de servicio de interrupción (vector de interrupción).
6. Se salva en la pila el estado del procesador.
7. La dirección de la rutina de servicio de interrupción se carga en el contador de programa, transfiriéndose el control.
8. La ejecución continúa hasta que el procesador encuentre la instrucción de retorno.
9. El procesador restaura el estado salvado anteriormente en la pila, devolviéndose el control al programa interrumpido.

Cuestión 5

Cada proceso está representado en el S.O. por el Bloque de Control de Proceso (PCB), una estructura de datos que contiene la información asociada a un proceso específico. La estructura del PCB de cada proceso se almacena en la zona de memoria reservada al sistema operativo.

- Estado del proceso: nuevo, preparado, en ejecución, esperando o detenido.
- El contador de programa: dirección de la siguiente instrucción a ejecutar.
- Los registros de la CPU. Esta información permite reanudar la ejecución del proceso.
- Información de gestión de memoria: qué zona de la memoria está siendo ocupada por el proceso.
- Lista de ficheros abiertos por el proceso e información sobre E/S pendientes.
- Información de contabilidad: tiempo real y de CPU empleados, límites de tiempo, identificador del proceso, etc.
- Información sobre la planificación de la CPU: prioridad del proceso, punteros a su situación en las colas, etcétera.

El PCB se actualiza en las siguientes situaciones: cuando el proceso cambia de estado, cuando entra o sale de la CPU, cuando se modifica su situación en alguna cola, cuando solicita alguna operación de E/S, o cuando se modifica su situación en memoria.

Cuestión 6

El mecanismo de paginación busca evitar las limitaciones de los recubrimientos, automatizando su funcionamiento. El conjunto de direcciones accesibles se divide en páginas, y esas páginas se pueden cargar en cualquier zona de memoria. La ventaja de este sistema es que permite enviar al disco las páginas que menos se usen: mecanismos de *memoria virtual*.

El mecanismo de paginación utiliza dos tipos de direcciones: las *direcciones virtuales*, que es el conjunto de direcciones que el programa puede utilizar, y las *direcciones físicas*, que son las direcciones de memoria física del sistema. La memoria física se dividirá en "marcos de página", destinadas a almacenar las páginas virtuales. Para localizar las páginas que se pretenden acceder, el sistema mantiene una tabla denominada *tabla de páginas*. Esta tabla indica dónde está cada página en cada momento: si en memoria, en disco o no existe.

Una dirección virtual se divide en dos partes: etiqueta y desplazamiento dentro de la página. Para localizar físicamente cada página se utiliza la etiqueta como índice de la tabla de páginas. Una vez cargada la página en memoria, el desplazamiento indica la localización del dato concreto. Cada vez que se intenta acceder a una dirección virtual, se comprueba la localización de la página. Si no está en memoria, se la trae, desalojando otra. Esta tarea la realiza un módulo especial, el MMU (*memory management unit*). Por lo tanto, para usar paginación debe existir hardware de apoyo en la CPU.

El funcionamiento de los mecanismos de paginación es transparente al programador: éste sólo usa la memoria, independientemente de la cantidad de memoria física que tenga instalada.