INTRODUCCIÓN A LOS COMPUTADORES

2003-2004

Objetivos Formativos

Que el alumno sea capaz de:

- Comprender el funcionamiento y saber diseñar los circuitos digitales combinacionales y secuenciales que se utilizan en el diseño del procesador pedagógico SISP-1-1 y SISP-1-2[1].
- Comprender el lenguaje máquina pedagógico SISA-1, la estructura interna, a nivel de circuitos digitales, de los procesadores SISP-1-1 y SISP-1-2, y como estos procesadores ejecutan las instrucciones SISA-1.

1. Introducción

1.1. Definir conceptos básicos tales como señal eléctrica digital y codificación de la información.

2. Números naturales: sistema de numeración y operaciones básicas

Comprender el sistema convencional en base b para la representación de números naturales y saber como se realizan las operaciones aritméticas básicas en binario.

2.1. Expresar el valor de un número en función de los dígitos que lo representan en el sistema convencional en base b.

2.2. Expresar el rango de la representación en función de la base y del número de dígitos.

2.3. Dado un número representado en base 2 (binario), base 16 (hexadecimal) o base 10 (decimal), obtener su representación en otra de esas bases.

2.4. Dado un número representado en base 2 (binario), base 16 (hexadecimal) o base 10 (decimal) con n dígitos, obtener su representación en la misma base pero con n+1 dígitos (algoritmo de extensión de rango).

[1] En el procesador SISP-1-1 cada instrucción tarda un ciclo en ejecutarse mientras que en el SISP-1-2 cada instrucción tarda varios ciclos en ejecutarse y unas instrucciones tardan más ciclos que otras. Estos procesadores se construyen a partir de puertas NOT, AND y OR y biestables D activados por flanco. No es necesario saber como las puertas y los biestables están construidos internamente, sólo debe conocerse su funcionalidad.

2.5. Efectuar las siguientes operaciones aritméticas básicas con números representados en binario con n bits ($n \leq 16$), incluida la detección de resultado no representable en n bits: suma, resta, multiplicación por potencias de 2, división por potencias de 2 y multiplicación.

3. Introducción a los circuitos lógicos combinacionales

Comprender qué es un circuito lógico combinacional y ser capaz de implementar circuitos de pocas entradas a partir de puertas NOT, AND y OR o de una memoria ROM.

3.1 Definir sistema lógico combinacional.

3.2. Especificar sistemas combinacionales pequeños[2]. Obtener la tabla de verdad de un sistema combinacional a partir de un texto que describe su funcionamiento.

3.3. Conocer el símbolo, la tabla de verdad y la operación lógica que implementan las puertas NOT, AND y OR (las puertas AND y OR con 2 o más entradas).

3.4. A partir de una tabla de verdad de un sistema combinacional pequeño, dibujar un circuito que sea su implementación directa a 2 niveles en suma de minterms (NOT-AND-OR).

3.5. Conocer el símbolo, la tabla de verdad, una expresión lógica de cada una de las salidas y la implementación con puertas NOT, AND y OR de: la puerta XOR de 2 entradas, el decodificador 1-2 y 2-4 (con entrada de habilitación) y el multiplexor 2-1. Describir con palabras propias la funcionalidad de cada uno de estos elementos combinacionales para el caso general de n entradas.

3.6. Obtener la tabla de verdad de un sistema combinacional pequeño a partir de su esquema. El esquema consiste en la interconexión de no más de 6 de los siguientes elementos: puertas NOT, AND, OR y XOR y/o los bloques combinacionales decodificador 1-2 y 2-4 y multiplexor 2-1.

3.7. A partir de la tabla de verdad de un sistema combinacional pequeño y de una lista con algunos axiomas y teoremas del álgebra de Booloe, obtener varias implementaciones con puertas NOT, AND y OR, mediante la manipulación de expresiones lógicas. De entre los diseños obtenidos, saber elegir los requieren menor hardware y/o menor tiempo de propagación.

3.8. Obtener el cronograma de las señales de salida de un sistema combinacional pequeño a partir de su esquema formado por puertas NOT, AND y OR, el retardo de cada puerta y la evolución temporal de las señales de entrada.

3.9. A partir de una tabla de verdad de un sistema combinacional pequeño, obtener la implementación con un decodificador y puertas OR.

---

[2] Consideramos que un sistema combinacional es pequeño si número_bits_entrada ≤ 4.

3.10. Describir con palabras propias la funcionalidad de una ROM.

3.11. Dibujar el esquema lógico interno de una ROM, mediante un decodificador, interconexiones y una puerta OR para cada salida.

3.12. Implementar sistemas combinacionales con una ROM. Relacionarlo con la implementación con un decodificador y puertas OR.

4. Introducción a los circuitos lógicos secuenciales

Comprender que es un circuito lógico secuencial y ser capaz de implementar circuitos con pocas entradas y pocos estados a partir de puertas NOT, AND y OR o memorias ROM y biestables D activados por flanco.

4.1. Justificar con ejemplos la necesidad de sincronización (reloj) para poder interpretar correctamente una secuencia de bits.

4.2. Justificar con ejemplos la necesidad de memoria en algunos sistemas digitales, que por lo tanto no pueden implementarse como combinacionales.

4.3. Definir sistema lógico secuencial.

4.4. Conocer el símbolo, el grafo de Moore y la funcionalidad de un biestable D activado por flanco.

4.5. Dibujar el esquema de un secuencial mediante los bloques: combinacional de estado siguiente, biestables de estado y combinacional de salida.

4.6. Especificar sistemas secuenciales pequeños[3]. Obtener el grafo de Moore de un sistema secuencial a partir de un texto que describe su funcionamiento.

4.7. Sintetizar sistemas secuenciales pequeños. A partir del grafo de estados de Moore encontrar un circuito que lo implemente:

4.7.1. Con un biestable por estado, decodificadores (demultiplexores) y puertas OR.

4.7.2. Con el mínimo número de biestables D y alguna de las técnicas de síntesis de combinacionales (con puertas en suma de minterms, con decodificador y puertas OR o con ROM).

4.8. A partir del esquema de un circuito secuencial pequeño de Moore formado por puertas NOT, AND y OR y biestables D, obtener el tiempo de ciclo mínimo del sistema (dados los tiempos de propagación de las puertas).

4.9. A partir del esquema de un circuito secuencial pequeño de Moore formado por puertas NOT, AND y OR y biestables D, obtener el grafo de estados de Moore.

[3] Consideramos que un sistema secuencial es pequeño si número_bits_entrada + log2(número_estados)≤ 4.

5. Procesadores de propósito especifico

Comprender el funcionamiento y saber diseñar procesadores de propósito específico que manipulan palabras de n bits (v.g. n=16) formados por Unidad de Proceso y Unidad de Control. La unidad de proceso se construye mediante un diseño ad-hoc con bloques combinacionales y secuenciales y la unidad de control se especifica mediante un grafo de estados.

5.1. Justificar con ejemplos la inviabilidad de especificar sistemas digitales que procesan palabras de n bits (por ejemplo para n=16 o mayor) mediante tablas de verdad y grafos de estados.

5.2. Justificar con ejemplos la viabilidad de diseñar sistemas digitales que procesan palabras de n bits mediante la descomposición del sistema en unidad de proceso y unidad de control, la realización de un diseño estructurado a nivel de bloques ad-hoc para la unidad de proceso y la realización de la unidad de control mediante alguna de las técnicas del objetivo 4.7. ¿Que señales se procesan en la unidad de control y cuáles en la unidad de proceso?

5.3. Conocer el símbolo, la tabla de verdad, una expresión lógica de cada una de las salidas y la implementación con puertas NOT, AND y OR del Half-adder y del Full-adder. Saber describir con palabras propias su funcionalidad.

5.4. Implementar multiplexores 4-1, 8-1 y 16-1 a partir de multiplexores 2-1.

5.4. Implementar los decodificadores 2-4, 3-8 y 4-16 a partir de decodificadores 1-2.

5.5. Implementar bloques combinacionales que procesen palabras de n bits, interconectando bloques combinacionales de menos entradas.(Los bloques resultantes serán usados en las unidades de proceso).

5.5.1. Multiplexor de buses 2-1, a partir de multiplexores 2-1.

5.5.2. Multiplexores de buses 4-1, 8-1 y 16-1 a partir de multiplexores de buses 2-1.

5.5.3. Incrementador binario, A+1, para A natural codificado en binario con n bits, a partir de Half-Adders.

5.5.4. Sumador binario, A+B para A y B naturales codificados en binario con n bits, a partir de Full-Adders con propagación del acarreo. Señal de resultado no representable en n bits a partir de puertas NOT, AND y OR.

5.5.5. Restador binario, A-B para A y B naturales codificados en binario con n bits, a partir de Full-Adders y puertas NOT. Señal de resultado no representable en n bits a partir de puertas NOT, AND y OR.

5.5.6. Operadores lógicos bit wise NOT, AND y OR, a partir de puertas lógicas NOT, AND y OR, respectivamente.

5.5.7. ALU con funcionalidades A+B, A-B, LSR(A), A AND B, A OR B, NOT(A) y A, a partir de los bloques aritméticos y lógicos de n bits correspondientes y un multiplexor de buses 8-1.

5.5.8. Comparador de números naturales, CMP, con funcionalidades LTU, LEU, GTU y GEU, a partir de un restador de dos números de n bits con salidas de carry y cero, puertas lógicas y un multiplexor 4-1.

5.6. Implementar bloques secuenciales de palabras de n bits a partir de biestables D activados por flanco y bloques combinacionales.(Los bloques resultantes serán usados en las unidades de proceso).

5.6.1. Registro de n bits a partir de biestables D activados por flanco.

5.6.2. Registro de n bits con control de carga a partir de un registro de n bits y un multiplexor de buses 2-1.

5.6.3. Banco de 8 registros de n bits cada uno con uno o dos buses de lectura y 1 bus de escritura, a partir de un decodificador 3-8, 8 registros de n bits con control de carga y uno o dos multiplexores de buses 8-1.

5.7. Conocer el protocolo de comunicación asíncrona: Four-Cycle Handshaking (o return-to-zero signaling).

5.7.1. Dibujar un cronograma con las señales involucradas en una entrada o una salida de datos con comunicación asíncrona. Expresar con palabras propias como funciona el protocolo en una entrada o salida de datos y porqué se efectúa correctamente independientemente de las velocidades del procesador y del dispositivo de entrada/salida.

5.7.2. Dibujar la parte del grafo de estados de la unidad de control del procesador de propósito específico que efectúa una entrada de datos. Lo mismo para una salida de datos.

5.8. A partir de una descripción textual de la funcionalidad de un procesador de propósito específico de n bits, encontrar una implementación formada por Unidad de Proceso y Unidad de Control, utilizando para la unidad de proceso un diseño ad-hoc con bloques digitales como los descritos en objetivos anteriores y especificando la unidad de control mediante un grafo de estados. La unidad de proceso no tendrá más de 8 bloques y la unidad de control no más de 10 estados.

5.9. A partir de un procesador de propósito específico de n bits formado por unidad de proceso (construida con no más de 8 bloques digitales) y unidad de control (especificada mediante un grafo de Moore de no más de 10 estados), obtener una descripción textual de su funcionalidad y el tiempo de ciclo mínimo (conocidos los tiempos de propagación de los bloques).

6. Añadiendo números enteros: sistemas de numeración, operaciones básicas e implementación

Comprender los sistemas de representación de números enteros: “signo y magnitud” en decimal y binario y “complemento a 2”. Saber como se realizan las operaciones aritméticas básicas en complemento a 2.

6.1. Conocer la expresión aritmética que nos da el valor de un número en función de los dígitos que lo representan en complemento a 2.

6.2. Conocer el rango de la representación en complemento a 2.

6.3. Obtener la representación de un número en complemento a 2 a partir de la representación en signo y magnitud decimal y viceversa.

6.4. Saber efectuar operaciones aritméticas básicas con números representados en complemento a 2, incluida la detección de resultado no representable: suma, resta, multiplicación por potencias de 2 y división por potencias de 2.

6.5. Justificar las ventajas de la representación complemento a 2 frente a signo y magnitud.

6.6. Implementar las operaciones aritméticas básicas del objetivo 6.4.

6.7. Implementar una ALU con funcionalidades A+B, A-B, ASR(A), A AND B, A OR B, NOT(A) y A, a partir de los bloques aritméticos y lógicos de n bits correspondientes y un multiplexor de buses 8-1.

6.7. Implementar un comparador de números enteros, CMP, con funcionalidades LT, LE, GT y GE, a partir de un restador de dos números de n bits con salidas de overflow, negativo y cero, puertas lógicas y un multiplexor 4-1.

7. Hacia el procesador de propósito general

Comprender las similitudes y diferencias entre un procesador de propósito específico y uno de propósito general y saber justificar: la conveniencia del secuenciamiento implícito, la codificación de las instrucciones mediante un formato compacto y la existencia de una memoria de datos junto con sus instrucciones de acceso load y store.

De la unidad de proceso de propósito específico a la de propósito general

7.1. Dada una unidad de proceso general (formada por un banco de registros con dos buses de lectura y uno de escritura, una ALU y un multiplexor en el bus de escritura del banco de registros para seleccionar la escritura desde la entrada de datos o desde la ALU) y conocida la palabra de control que la gobierna, dibujar el grafo de estados de Moore de la unidad de control para realizar una funcionalidad descrita mediante un texto.

De la unidad de control de propósito específico a la de propósito general y el programa

7.2. Idem al objetivo anterior, pero ahora las señales de salida que genera la unidad de control en cada ciclo se especifican mediante los mnemotécnicos previamente definidos para cada uno de los campos de la palabra de control y para el secuenciamiento explícito (siguiente estado) -por ejemplo: ADD R1, R3, R5 | go to 6.

7.3. Razonar las ventajas e inconvenientes del secuenciamiento implícito frente al explícito.

7.4. Razonar las ventajas e inconvenientes de la codificación de las instrucciones con un formato compacto frente a tener todos los campos de la palabra de control con funcionalidades distintas.

7.5. Dada la unidad de control general UCG0(formada por un registro contador de programa, un incrementador, el secuenciador y la ROM de programa) y la unidad de proceso UPG0, escribir el contenido de la ROM con mnemotécnicos.

7.6. Justificar la conveniencia de introducir una memoria de datos y las instrucciones de load y store.

7.7. Justificar la conveniencia del modo de direccionamiento registro base mas desplazamiento en las instrucciones de load y store.

8. Lenguaje Máquina y Ensamblador

Conocer el juego de instrucciones del lenguaje máquina SISA-1 y su especificación en lenguaje ensamblador y los procesos involucrados en la ejecución de un programa escrito en un lenguaje de alto nivel.

8.1. Saber ensamblar y desensamblar código SISA-1 ayudados por una tabla que contiene el formato y codificación de las instrucciones.

8.2. Entender pequeños programas escritos en lenguaje ensamblador del SISA-1 (como máximo 10 instrucciones), indicando como queda modificado el estado del computador después de su ejecución.

8.3. Escribir pequeños programas en lenguaje ensamblador del SISA-1 (como máximo 10 instrucciones) cuya funcionalidad se especifica mediante un texto o mediante una sentencia sencilla de un lenguaje de alto nivel.

8.4. Describir en que consisten los procesos involucrados en la ejecución de un programa escrito en un lenguaje de alto nivel(compilación, ensamblado, carga y ejecución de las instrucciones en el hardware).

9. El procesador SISP-1-1 (uniciclo)

Comprender la estructura interna y el funcionamiento del procesador SISP-1-1, en el que cada instrucción tarda un ciclo en ejecutarse.

9.1. Dibujar el procesador SISP-1-1 a nivel de bloques y saber implementar cada bloque con bloques más pequeños hasta el nivel de puertas NOT, AND, OR y biestables D activados por flanco.

9.2. Indicar el valor de algunas señales de la Unidad de Proceso y de la Unidad de Control durante la ejecución de una instrucción.

9.3. Obtener el tiempo de ciclo mínimo (dados los tiempos de propagación de los circuitos combinacionales).

9.4. Elegir entre varias alternativas de diseño de la UP y/o UC, aquella que reduzca el tiempo de ciclo y/o la que requiera menor hardware, razonando la respuesta.

9.5. Proponer modificaciones del SISP-1-1 para que pueda ejecutar una nueva instrucción de lenguaje máquina (de complejidad equivalente a las originales) y/o para que la implementación cumpla restricciones respecto del hardware disponible, del tiempo de ciclo o del número de ciclos de ejecución de cada instrucción.

10. El procesador SISP-1-2 (multiciclo)

Comprender la estructura interna y el funcionamiento del procesador SISP-1-2, en el que cada instrucción tarda varios ciclos en ejecutarse y no todas tardan el mismo numero de ciclos.

10.1. Justificar las ventajas de una implementación multiciclo (SISP-1-1 v.s. SISP-1-2).

10.2. Dibujar la unidad de proceso del SISP-1-2 a nivel de bloques, resaltando y sabiendo justificar la razón de las diferencias con el SISP-1-1.

10.3. Dibujar el grafo de estados de la unidad de control indicando el valor de todas las transiciones y salidas.

10.4. Indicar la evolución temporal (cronograma) de algunas señales de la Unidad de Proceso y de la Unidad de Control durante los ciclos que dura la ejecución de alguna instrucción de lenguaje máquina (o una secuencia de dos o tres instrucciones).

10.5. Obtener el tiempo de ciclo mínimo (dados los tiempos de propagación de los circuitos combinacionales), el numero de ciclos en que se ejecuta cada instrucción y el tiempo de ejecución de un programa (dado el número de instrucciones ejecutadas de cada tipo).

10.6. Elegir entre varias alternativas de diseño de la UP y/o UC, aquella que reduzca el tiempo de ciclo y/o la que requiera menor hardware y razonar la respuesta.

10.7. Proponer modificaciones del procesador SISP-1-2 para que pueda ejecutar una nueva instrucción de lenguaje máquina (de complejidad equivalente a las originales) y/o para que la implementación cumpla restricciones respecto del hardware disponible, del tiempo de ciclo o del número de ciclos de ejecución de cada instrucción.