CAPÍTULO 4 - La arquitectura de programación

EL lenguaje de máquina se analiza en función del llamado lenguaje ensamblador o lenguaje simbólico, funcionalmente equivalente al lenguaje de máquina correspondiente nombre en reemplazo de palabras binarias.

Bus: su propósito es reducir la cantidad de intercomunicaciones entre la cpu y sus subsistemas. En lugar de tener caminos de comunicación diferentes con la máquina y con cada uno de los dispositivos de entrada-salida, la CPU se interconecta con la memoria y con con los sistemas de entrada y salida a través del bus del sistema.
CPU (alu, registros y unidades de control), memoria y dispositivos de entrada y salida.

No todos los componentes del sistema se conectan al bus de la misma forma.
La CPU genera direcciones que se transfieren sobre el bus de direcciones, mientras que la memoria recibe esas direcciones a través de bus de direcciones. La memoria nunca genera direcciones, así como la cpu nunca recibe direcciones.

PASOS:

- Usuario escribe un programa de alto nivel
- Se traduce a lenguaje ensamblador por medio del compilador
- Un programa ensamblador convierte el programa en lenguaje simbólico de máquina, para su almacenamiento en el disco
- Como paso previo a su ejecución, el sistema operativo de la computadora carga el programa en lenguaje de máquina desde el disco a la memoria principal
- Durante su ejecución cada instrucción se carga en el CPU desde la memoria, junto con cualquier dato que sea necesario para ejecutar la instrucción
- La salida del programa se coloca en un dispositivo
- **TODAS LAS INSTRUCCIONES ESTÁN REGULADAS POR LA UNIDAD DE CONTROL**

La comunicación de los 3 componentes se maneja por medio de buses
Todas las instrucciones se ejecutan dentro del CPU a pesar de que las instrucciones y los datos se encuentran almacenados en memoria. Esto significa que las instrucciones deben cargarse desde la memoria principal hacia los registros de la CPU, en tanto que los resultados deben devolverse a la memoria para su almacenamiento desde los registros de la CPU.

Memoria
Consiste en una serie de registros numerados (direccionados) en forma consecutiva, cada uno de los cuales almacena un byte de información (byte 8 bits). Cada registro tiene una dirección a la que suele designar como LOCALIZACIÓN DE MEMORIA. NIBBLE 4 bits

Cuando se utilizan palabras de más de un byte se puede usar BIG ENDIAN: byte más significativo en la dirección las baja de memoria y LITTLE ENDIAN byte menos significativo se almacena en la posición más baja de memoria.

La estructura de la memoria consiste en un arreglo lineal de las diversas localizaciones ordenadas en forma consecutiva. Las direcciones son un número que identifica cada palabra y se cuentan en secuencia a partir de cero y tiene un **espacio de direcciones** de 32 bits. La última dirección corresponde a 2^32 -1. El espacio de direcciones de la arquitectura está dividido en diferentes sectores, los que se utilizan para el sistema operativo, para los elementos de entrada salida, para los programas del usuario y para la paila del sistema. A la distribución de estos sectores se lo llama **mapa de memoria**.
Las primeras 2^11 = 2048 se reservan para el sistema operativo. El espacio del usuario es el que se reserva para la carga de los programas ensamblados por el usuario y puede crecer, durante la operación, desde la dirección 2048 hasta que se encuentre la pila del sistema. La pila del sistema comienza en la dirección 2^-31 - 4. La zona de espacio para direcciones ubicadas entre 2^31 y 2^32-1 se reserva para la dispositivos de entrada y salida.

“El mapa de memoria no está compuesto enteramente por memoria real y de hecho pueden existir espacios en los que no haya ni memoria ni direcciones de dispositivos de entrada salida. Dado que los dispositivos e-s se tratan de

posiciones de memoria, la lectura y escritura de dichos dispositivos se puede realizar a través de los mismos comandos de lectura y escritura utilizados para la memoria del sistema. ENTRADA Y SALIDA MAPEADA EN MEMORIA”

Diferencia entre dirección y dato: ej en que ambas tienen 32 bits. Una dirección es un puntero a una posición de memoria, lo cual tiene un dato.
Espacio de direcciones se refiere al rango numérico de direcciones de memoria al que se puede hacer referencia la unidad del proceso. Una palabra de direcciones formada por n bits puede especificar una de 2^n direcciones (espacio de dir 2^n bytes).

UNIDAD DE PROCESO DE LA CPU

Registros contador de programa **(pc,** program counter) y registro de instrucción **(ir)** forman la interfaz entre la unidad de control y la unidad de datos.
El pc tiene la dirección de la instrucción en ejecución. la instrucción a la que apunta el PC se rescata de memoria y se almacena en el IR, desde donde se interpreta

TRAYECTO DE DATOS	**UNIDAD DE CONTROL**
registros y unidad aritmetico logica (ALU) ->	Interpreta las instrucciones y realiza la transferencia entre registros
Registro: memoria pequeña y rápida, separada de la memoria del sistema que se usa para el almacenamiento temporario durante las operaciones de cálculo. Cada registro recibe una dirección, mucho más chica que las de memoria: 32 registros -> palabra de direcciones de 5 bits. Está contenido en la CPU. Instrucción sobre operandos en registros es 10+ rapido que si estuvieran en memoria aunque hace falta más operaciones.	La responsable de la ejecución de las instrucciones del programa, las que se almacenan en la memoria principal. Coordina las distintas unidades que vienen en la ejecución de un programa de computadora Toma decisiones acerca del comportamiento del resto de la máquina
3 buses conectan el trayecto de datos con el bus del sistema para la transferencia de datos entre memoria principal y registros. 3 buses conectan registros con ALU y permiten la búsqueda simultánea de 2 operandos de registros para ser procesados por la ALU, devuelve el resultado al conjunto de registros. 2 operandos se transfieren desde registros a través de los buses que se conectan los los registros de origen (rs1, rs2). La salida de la ALU se coloca en el bus que conduce al registro destino (rd).	Los pasos que lleva la unidad de control: 1. Búsqueda en memoria de la próxima instrucción ser ejecutada 2. Decodificación del código de operación 3. Búsqueda de operandos en memoria, si los hubiera 4. Ejecución de la instrucción y almacenamiento de los resultados 5. Vuelta al paso 1
Incluyen la conexión con el bus del sistema para acceder a memoria y a e-s (bus datos bus direcciones)	Selecciona las operaciones y los operandos a utilizar

La unidad de control

Compilador:
Programa de computadora que transforma programas escritos un lenguaje de alto nivel en lenguaje de máquina (lenguaje simbólico: assembly language)

Salida del compilador es la parte del programa responsable de la generación de código máquina para un procesador específico.

Proceso de compilación **(proceso de traducción**) y luego, Programa ensamblador (assembler) traduce el lenguaje simbólico hacia el código de máquina del procesador de destino.

Estas traducciones se producen en los **momentos de compilación y momento de ensamble.**

El programa objeto resultante puede vincularse con otros programas objeto en el **momento del enlace (link).** El programa objeto, normalmente almacenado en un disco, se carga en la memoria principal en el **momento de la carga (load**) y se ejecuta desde la CPU en el **momento de la ejecución (run)**

SPARC - ARC -RISC (conjunto reducido de un conjunto de instrucciones)
ARC: Máquina de 32 bits con capacidad de almacenamiento por bytes, pueden manejar tipos de datos de hasta 32 bits. Cada entero se almacena en memoria como un conjunto de 4 bytes con big endian
"la máxima dirección disponible para el almacenamiento es 2^31 -1 de modo que la dirección de la palabra más alta de memoria está ubicada 3 bytes por debajo de esta en 2^31 -4

CPU:

- 32 reg de 32 bits. pc y ir.
- registro de estado del procesador(PSR, precessor status registrer) ej resultado de op aritméticas. Los flags (códigos de condición)
- Tamaño de todas las instrucciones son de una palabra (32 bits)
- ARC es una máquina de arquitectura carga-descarga (load-store) Las únicas instrucciones permitidas para el acceso de memoria cargan el valor hacia alguno de los registros. Todas las operaciones aritméticas se ejecutan sobre operandos contenidos en registros y los resultados también quedan almacenados en un registro
- Registros %r14 (%sp puntero a pila) y %r15 (registro de enlace, link registrer)

Instrucciones ARC

- SETHI: carga los 22 bits más significativos en un registro. se usa con otra instrucción que carga los 10 bits menos significativos.
- andcc, orcc, orncc: Uno de sus operandos debe estar en un registro. El otro puede estar en un registro o puede ser una constante incluida en la instrucción, expresada en 13 bits en notación de complemento a dos, la que se extiende a 32 bits cuando se la utiliza. El resultado se almacena en un registro. Los sufijos "cc" es para que los códigos de condición se actualicen
- srl: desplazamiento lógico a derecha (0s en bits +sig)
- sra: desplazamiento aritmético a derecha (1 o 0s en bits +sig)

FORMATO DE INSTRUCCIONES EN ARC:
Los 2 bits más significativos forman un campo **op** correspondiente al código de operación. campo de 5 bits de **rd** identifica el registro al que se aplicará la instrucción. **op2** tipo de operación.
campo con identifica el tipo de salto condicional, que se basa en los códigos de condición del registro de estado PSR.
Reg origen rs1, rs2. El campo **op3**: código de operación que identifica la instrucción.
Cuando el campo i = 1 el campo simm13 (13 bits) se extiende a 32 bits con bit más significativo.
Cuando i = 0 los operandos contenidos en el campo rs1 y rs2 se suman para obtener la dirección. Cuando i = 1, la dirección se obtiene sumando los campos rs1 y simm13

Call invoca a una subrutina y almacena la dirección de la instrucción actual en %r15. La dirección de la siguiente instrucción a ser ejecutada se obtiene sumando 4*disp30 con la dirección de la instrucción en ejecución. Para el código objeto indicado a continuación la sub_r se encuentra ubicada en memoria 25 palabras (100 bits) después de la instrucción call
jmpl: salta y vincula (retorno a subrutina) salta a una nueva dirección y almacena la dirección de la instrucción actual en el registro de destino.

Directivas: específicas de un determinado ensamblador

- .equ igualar a un símbolo, .begin, .end, Es útil para el proceso de depuración porque permite que partes del programa se hagan invisibles al ensamblado. se ignora lo que está fuera.
- .org hace que la instrucción siguiente de ensamble, suponiendo que se la ubicara en el momento de la ejecución en la posición de memoria específica.
- .dwd reserva un bloque de palabras de 4 bytes generalmente para arreglos. El contador de posiciones se desplaza hasta ubicarlo delante del bloque, de acuerdo con la cantidad de posiciones obtenidas al multiplicar por 4 el argumento de la directiva, ya que dicho argumento especifica la cantidad de palabras necesarias

- .global hace accesible un rótulo para que pueda ser usado en otros módulos y .extern identifica un rótulo que está siendo usado en el módulo local y se ha definido en otro módulo (rótulo qUe en el otro módulo debería estar indicado como .global)

VARIANTES EN LAS ARQUITECTURAS Y EN LOS DIRECCIONAMIENTOS

la arquitectura de ARC es la típica arquitectura de una máquina de carga y descarga. Los programas se ejecutan en menos tiempo, en parte debido a la reduccion del trafico entre CPU y la memoria, porque los operandos se cargan una vez en la CPU y los resultados se salvan en memoria únicamente cuando se completa el cálculo.
El tráfico de memoria tienen dos componentes, por un lado la instrucción que debe traerse de memoria a la CPU para poder ser ejecutada y por otro los datos: operandos que deben transladarse a la CPU para realizar la operación y resultados que deben ser devueltos a la memoria cuando se completa el cálculo.

PILAS Y SUBRUTINAS

Subrutina: función o procedimiento, es una secuencia de instrucciones a la que se invoca. Cuando un programa llama a una subrutina, se transfiere el control del programa a la subrutina que luego vuelve a la posición siguiente a la que generó el llamado. Transferencia de elementos (enlace entre subrutinas) Convenciones de llamada para pasar son diferentes.

1) Argumentos en registros, sencillo pero no funciona si la cantidad de elementos excede el número de registros disponibles o si la llamada a subrutinas están fuertemente encadenadas.
2) Zona de transferencia de datos: se entrega datos en un registro predeterminado. La directiva .dwb genera una zona de transferencia de datos de X palabras, X corresponde a la dirección de comienzo de la zona de datos que se le transfiere a la subrutina en un registro. Se cargan los argumentos en la posición x, x+4 y la subrutina los toma de la dirección X y guarda en X+8. Permite el pasaje de bloques de cualquier tamaño sin tener que copiar más que un registro en el proceso de llamado a la subrutina. PERO puede complicarse si hay rutinas recursivas. Debido a que la rutina que se invoque a sí misma podría generar varias zonas de transferencias. Su tamaño debe ser conocido en tiempo de ensamble. Un caso alternativo es almacenar la dirección x en memoria para su posterior extracción, es más sencilla pero como involucra a llamados a memoria es más lenta.
3) Pilas a subrutinas. Rutina invocante coloca todos sus argumentos (o punteros si los datos tienen tamaño grande) a la pila. La rutina invocante extrae de la pila y luego coloca el valor de retorno. El programa invocante rescata de la pila los valores devueltos. El registro de la CPU conocido como puntero a pila tiene la dirección de la cabeza de la pila. La ventaja en su uso es que su espacio aumenta o disminuye respecto a su uso. Admite encadenar llamadas a procedimientos sin tener que declarar el tamaño en el momento de ensamble.

Si la rutina invocada llama a su vez a otra rutina, el valor del contador de programa que había sido guardado originalmente en r15 se verá sobrescrito por la llamada encadenada, lo que implica que no se podrá retomar correctamente al programa principal a través de %r15. Con el objetivo de permitir las llamadas con retornos encadenados el valor actual de %r15 (llamado registro de enlace) debería descartarse en la pila junto con cualquier otro registro que requiera ser ocupado luego del retorno.
Si se utiliza una convención de llamada basada en registros, antes de proceder a una llamada encadenada debería rescatarse el registro de enlace en alguno de los registros disponibles.
Si se usa una zona de transferencia de datos, la misma debería disponer de espacio reservado para que el registro de enlace.
Si se usa una pila, el registro de enlace también debe apilarse.
Para cada una de las convenciones utilizadas para la llamada a subrutinas, antes de encadenar subrutinas deben rescatarse los contenidos del registro de enlace y los valores locales. En caso contrario cuando se produzca una llamada encadenada a la misma subrutina se producirá la pérdida la pérdida de las variables locales.

La única memoria físicamente existente en el sistema ocupa el espacio de direcciones entre 2^22 y 2^23 -1 (2^23-4 es la dirección del byte más significativo de la palabra más alta en el formato de almacenamiento big endian). El área de direcciones que va desde 0 a 2^16 -1, inclusive, contiene programas residentes para las operaciones de arranque y encendido y rutinas gráficas básicas. El espacio entre 2^16 y 2^19 -1 se utiliza para ubicar dos módulos adicionales de memoria de video.
Solo se tendrá información válida disponible en esas direcciones cuandos los módulos de memoria están físicamente ubicados en el sistema.

El espacio 2^23 y 2^24 se utiliza para e-s

CAPÍTULO 5 - Lenguajes de la máquina

Ensamblado: traducción del programa escrito en lenguaje ensamblador a otro, funcionalmente equivalente, expresado en lenguaje de máquina.

Macroinstrucciones del lenguaje ensamblador.

Pasos de la compilación:

- Reconocer símbolos del lenguaje (constantes,, =,+) -> Analisis lexico
- Reconocer símbolos y que significa en diferentes expresiones (a = 20, asignacion) ->Analisis sintactico
- Análisis nombres, asociar variables a su posición en memoria almacenada en la Ejecución.
- Análisis de tipo ->análisis semántico.
- Asignación de acciones y generación de código: asociar las sentencias de programa con la secuencia apropiada de lenguaje ensamblador.

Especificación del mapeo:
El compilador debe conocer la arquitectura de programación del procesador y maquina en la que se desarrolla -> especificación de asignación del compilador. Ej cómo ubicar variables en recursos de la máquina.

Almacenamiento de variables en memoria
Variables globales: tienen direcciones conocidas en el momento de compilación
Locales: declaradas dentro de funciones, toman existencia cuando se ingresa a dicha función y al finalizar desaparecen. Se usa pilas para almacenamiento de variables temporarias.
El uso de %fp es para determinar el desplazamiento en un valor almacenado en la pila. Se accede a partir del Direccionamiento base.
Las variables almacenadas en pilas están asignadas a direcciones que no se conocen hasta su momento de ejecución. Sus direcciones de memoria en el momento de compilación sólo se reconocen respecto a %fp

Movimiento de datos: tipos de estructuras, arreglos
Instrucciones aritméticas
Sentencia GOTO: salto incondicional branch always
IF-ELSE: salto por distinto branch if equal
WHILE: evaluar la expresión si es cierta ejecutar y repetir hasta que la expresión evaluada se convierta en falsa.
DO-WHILE, FOR

Proceso de ensamblado: proceso lineal. Relación unívoca entre lenguaje simbólico y lenguaje de máquina

- Debe permitir especificar ubicación de variables y programas en el momento de ejecución.
- Ofrecer posibilidad de inicializar los valores de los datos en memoria antes de la ejecución del programa
- Proveer expresiones en el lenguaje de programación para todas las instrucciones del lenguaje de máquina y modos de direccionamiento. Traducir sentencias hacía binario
- Permitir uso de rótulos para identificar o representar direcciones y constantes.
- Indicación de inicio o fin.
- Permitir uso de variables en dos módulos ensamblados por separado.
- Proveer macro rutinas.

Ensamblado en dos pasadas

1) Determina las direcciones de todos los datos e instrucciones del programa y selecciona qué instrucciones del lenguaje de máquina debe generarse para cada instrucción del lenguaje simbólico. Las direcciones de los datos y de las instrucciones se determinan en el momento del ensamblador mediante un contador similar al contador del programa: contador de posiciones que lleva la dirección de la instrucción de la instrucción. (.org hace que el contador de ubiquen en esa dirección). El ensamblador realiza una tabla de símbolos con operaciones aritméticas e inserta operación en los rótulos o constantes.

2) Permite el uso de símbolos dentro de un programa con anterioridad sean definidos. Referencia previa. En la segunda pasada se podrá generar el código de máquina, insertando en los mismos los valores de los símbolos

Tareas finales del programa ensamblador
Luego de completar el proceso de traducción, el ensamblador debe agregarle al modulo traducido informacion adicional para el uso de los programas de enlace y carga:

- El nombre y tamaño del módulo. Deben especificarse el tamaño de segmentos, datos, pilas
- La dirección de símbolos de comienzo.
- Información acerca de dirección de símbolos globales y externos.
- Información acerca de las rutinas de biblioteca a las que el módulo hace referencia.
- Valores de cualquier constante que deba cargarse en memoria
- Información de reubicación. Cuando se invoca un programa de enlace, la mayoría de los módulos a vincular deben reubicarse a medida que se los concatena.

Ubicación de programas en memoria
Los programas se encuentran reubicados en memoria, en una dirección especificada por una directiva .org. Al programador no le interesa saber en qué posición de memoria se carga el programa, ya que puede ser compilado o ensamblado de forma separada. La mayoría de las direcciones se especifican como reubicables en memoria, excepto las direcciones de entrada y salida que pueden ser fijas.
La información de reubicación se incluye en un diccionario de reubicación, en el módulo ensamblado, para que sea utilizado por el editor de enlace y/o el cargador.

Enlace y carga

LINKING (Enlace)	LOADER (Carga)
Unir en un programa único distintos módulos que fueron ensamblado de formas separadas (llamados modulo objetos) un único programa, o módulo de carga.	traslado del programa a memoria y su preparación para ser ejecutado
El linker resuelve todas las referencias globales y externas y reubica las direcciones de los diferentes módulos.	El módulo de carga puede ser cargado en memoria por medio de un cargador, que también puede necesitar modificar direcciones si el programa se carga en una dirección distinta a la dirección de origen de carga usada por el linker.

Enlace (linking)
El programa de enlace debe:

- Referencias de direcciones externas a los módulos a medida que los vincula.
- Reubicar los módulos.
- Especificar el símbolo de comienzo del módulo de carga.

Resolución de referencias externas
Al resolver las referencias de las direcciones, el programa de enlace necesita distinguir los nombres de los símbolos locales de los nombres de los símbolos globales.
La directiva **.global** le indica al ensamblador que debe señalizar al símbolo como disponible para otros módulos objetos durante la fase de enlace. **.extern** identifica a un rótulo usado en un módulo pero que se encuentra definido en otro. **.equ** se usa durante el ensamblado, el que se habrá completado en el momento que se inicia el proceso de enlace.

Cada una de las instrucciones tiene una longitud de 4 bytes

Reubicación
Dos módulos con la misma dirección de comienzo no pueden ocupar la misma dirección de memoria. Si se ensamblan por separado no se conoce el conflicto de memoria durante el proceso de traducción.

Para resolver el problema, el ensamblador define como reubicables a aquellos símbolos que pueden admitir que sus direcciones sean modificadas durante el proceso de enlace. Las reubicaciones se realizan a través del linker, de modo tal que las direcciones reubicables se modifican en el mismo valor en el que se modificó la dirección de carga.
Existen direcciones absolutas que se mantienen sin modificación independiente del proceso de carga.
La información para reubicar se almacena en un diccionario en el archivo ensamblado, disponible para el linker.

Carga (loader)
Es el programa que se ubica el módulo de carga en la memoria principal.
Si en todo momento se tiene un único módulo de carga, el módulo funciona correctamente. Actualmente los sistemas operativos tienen muchos programas que residen en memoria. El programa cargador debe reubicar estos módulos en el momento de carga, para lo cual se le sumará un desplazamiento de todo el código reubicable en un módulo dado ->**Cargador reubicador**.

El programa de enlace debe combinar varios módulos objeto en un único módulo de carga, mientras que el cargador simplemente modifica las direcciones reubicables que encuentra dentro de un módulo de carga dado para permitir la coexistencia en memoria de varios programas en forma simultánea. Un programa **cargador y de enlace** ejecuta tanto el proceso de enlace como el de carga: resuelve referencias externas, reubica módulos objeto y carga en memoria.
El archivo ejecutable contiene un encabezado con información que describe donde se lo debe cargar, cuál es su dirección de comienzo y si es posible que incluya información de reubicación así como puntos de entrada para cualquier rutina que deba estar disponible hacia el exterior del mismo

Una aproximación alternativa, que se basa en las técnica s de administración de memoria, realiza la reubicación por medio de la carga, en un registro base de segmento, de la base apropiada para la ubicación del código en el correspondiente lugar de la memoria física. La unidad de administración de memoria (**MMU**) suma el contenido de este registro base a todas las referencias de memoria. Como resultado, cada programa puede iniciar su ejecución en la dirección 0 y confiar en la MMU para lograr la reubicación de todas las referencias de memoria en forma transparente.

Macroinstrucciones…

CAPÍTULO 6- Trayecto de datos y control

Arquitectura de programación: un conjunto de instrucciones que realiza operaciones sobre los registros y sobre la memoria.
Unidad de control…
La microarquitectura está construida por la unidad de control y los registros accesibles al programador, las unidades funcionales del tipo de la unidad aritmetico logica y todo otro registro adicional que la unidad de control pueda requerir.
Una arquitectura de programación determinada puede implementarse con microarquitecturas diferentes.
Ej algunas microarquitecturas pueden apuntar a mejorar la velocidad de ejecución del juego de instrucciones, otras pueden apuntas a disminuir el consumo de energía y otras disminuir el costo del procesador. La posibilidad de modificar la microarquitectura sin modificar la arquitectura de programación implica que los fabricantes pueden aprovechar las nuevas tecnologías de fabricación de memorias y de circuitos integrados, para ofrecer a los usuarios compatibilidad en los desarrollos de software.

Fundamentos de la microarquitectura
La funcionalidad de una microarquitectura se centró en el ciclo de búsqueda y ejecución de una instrucción. Ciclo de búsqueda y ejecución:

1. Buscar en memoria la siguiente instrucción que debe ejecutarse.
2. Decodificar el código de la operación
3. Ejecutar la instrucción y almacenar los resultados
4. Volver al paso 1

La microarquitectura de una máquina incluye una sección de datos, que contiene registros y una unidad aritmético-lógica, y una sección de control. La sección de datos se conoce habitualmente como **trayecto de datos.** El control microprogramado utiliza un **microprograma**, no visible al usuario, el que implementa las operaciones sobre los registros y sobre otros sectores de la máquina. A menudo, el microprograma contiene muchos pasos de programa que

en su conjunto implementan una única (macro)instrucción. La unidad de control cableadas adoptan un enfoque el de generar distintos pasos requeridos para implementar una operación como estados sucesivos de una máquina de estados finitos.

Una microarquitectura para ARC

Trayecto de datos de ARC: contiene 32 registros (%r0-31), el contador del programa que apunta a la dirección a ser leída desde la memoria principal(%pc) , el registro de instrucciones de ejecución (%ir), la unidad arimetrico-logica (ALU), cuatro registros temporarios no accesibles al nivel de la arquitectura de programación que se utilizan para interpretar el conjunto de instrucciones de ARC (%temp0-3) y las conexiones entre estos elementos. El usuario tiene acceso directo al contador de programa solo a través de las instrucciones call y jmpl.

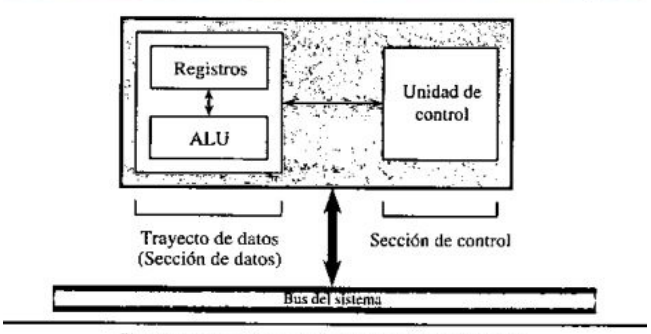


Figura 6.1 • Microarquitectura vista desde el alto nivel.

La unidad aritmetico-logica

Puede realizar una de 16 operaciones sobre los buses A y B. Cada operación realizada en la ALU, el resultado de 32 bits se coloca en el bus C, al menos que quede bloqueada por el multiplexor que maneja el bus C como consecuencia de colocar en él una palabra proveniente de la memoria de la máquina.

- ANDCC y AND: producto lógico de los operandos sobre los buses A y B. Si termina en CC afectan a los códigos de condición, afectan el contenido del registro de estado.
- ORCC y OR suma logica
- ADDCC y ADD suma aritmética, utilizando la representación de complemento y de los signos de los mismos.
- **SRL** (desplazamiento lógico a derecha) desplaza el contenido del bus A hacia la derecha en la cantidad de bits especificados por el dato del bus B (desde 0 hasta 31 desplazamientos). Se introduce ceros en los bits más significativos del resultado desplazado, en tanto que los bits menos significativos se descartan.
- **LSHIFT2 y LSHIFT10** desplazan a la izquierda el contenido del bus A, completando con ceros los bits menos significativos.
- **SIMM13** recupera los 13 bits menos significativos de la palabra contenida en el bus A y coloca ceros de los 19 bits restantes
- **SEXT13** realiza la extensión de signo de los 13 bits menos significativos de la palabra contenida sobre el bus A para crear una palabra de 32 bits
- INC incrementa el valor contenido en el bus A en 1
- **INCPC** incrementa el valor del bus A en 4, lo que utiliza para aumentar el contenido del contador del programa en una palabra de 4 bytes. Puede utilizarse sobre el contenido de cualquier registro contenido con el bus A.
- **RSHIFT5** desplaza 5 lugares hacia la derecha al operando que se encuentra en el bus A, copiando en los 5 bits libres de la izquierda el bit más significativo. Al aplicarse 3 veces consecutivas sobre una instrucción de 32 bits, la operación coloca el bit más significativo del campo **COND** del formato de saltos en la posición del bit 13. Es útil para la decodificación de instrucciones de salto.

Para pasar datos a través de la ALU sin modificarlos suele hacerse un producto lógico consigo mismo y el descarte a través del registro %r0.

La ALU genera los códigos de condición c, n, z y v. Se modifican a través de instrucciones y se genera una señal (**SCC**) que le indican al registro %psr que debe actualizar sus códigos de condición.

La alu tiene una tabla de 32 operaciones, seguida por un circuito de controlador de desplazamientos que implementa los desplazamientos. Los desplazamientos se producen por niveles y en cada nivel se observa un bit diferente de la entrada que indica la cantidad de desplazamientos.
Los bits N y C se obtiene directamente del la salida c31. El bit z se determina a partir de una NOR ejecutada sobre las salidas del circuito de desplazamiento. El bit v, vale 1 si el arrastre ingresado a la posición más significativa difiere del arrastre generado desde dicha posición, con una XOR.

Los registros
Implementados por biestables D activados por flanco negativo.Todos son de 32 bits de ancho. La entrada CLK al registro %r1 se introduce en una compuerta AND junto con la línea de selección correspondiente (c1) del decodificador C. esto asegura que la %r1 solo cambia cuando la selección de control lo determina. La entrada de datos de %r1 se toman directamente de las líneas correspondientes desde el bus C. Las salidas se escriben sobre el bus A y B, a través de compuertas buffer de 3 estados, las que se encuentran desconectadas eléctricamente salvo cuando sus entradas habilitan el valor. Las salidas de los buffers se transfieren hacia los buses A y B por medio de las salidas a1 y b1 de los decodificadores A y B. Si ni a1 ni b1 están en 1 se desconecta eléctricamente los buses.
Los demás registros adoptan un esquema similar salvo r0 que no tiene entradas desde C ni decodificador, no necesita flips flops.
El registro %ir tiene salidas adicionales que corresponden a los campos rd, rs1, rs2, op, op2, op3, y el bit 13. La unidad de control usa esas salidas en las interpretación de instrucciones.

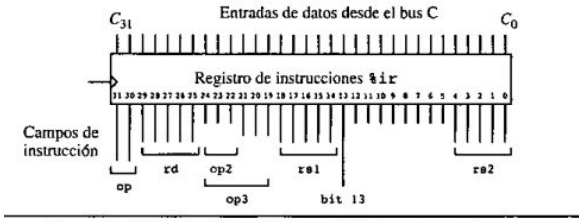


Figura 6.9 • Salidas hacia la unidad de control desde el registro %ir.

El contador del programa solo puede contener múltiplos de 4 por los que los bits menos significativos de %pc se conectan a 0.
Los decodificadores A, B y C simplifican la selección de registros. Las entradas de los decodificadores, de 6 bits, selecciona un único registro para cada uno de los buses A, B y C. Existen 2^6 = 64 posibles salidas desde los decodificadores, pero solo hay 38 registros de datos. La salida 0 del decodificador C no se usa porque %r0 no puede ser escrito. Los indicadores mayores a 37 no es para ningún registro y puede usarse cuando no se requiere conectar registro con alguno de los buses.

La selección de control
En el corazón de la unidad de control, una memoria (ROM) de 2048 palabras de 41 bits contiene los valores de todas las líneas que deben controlarse para implementar cada instrucción a nivel usuario. La memoria de lectura suele considerarse como una memoria de control. Cada palabra de 41 bits es una **microinstrucción.**
La unidad de control es la responsable de la búsqueda de las microinstrucciones y de su ejecución. La ejecución de una microinstrucción se controla a través del registro de instrucciones de microprograma (**MIR**), del registro de estado %psr y de un mecanismo que permite determinar cuál es la siguiente microinstrucción a ejecutar, formado por la unidad de saltos de control (CBL, control branch logic) y el multiplexor de direcciones de la memoria de control. No se requiere contador de de programa para almacenar la dirección de la próxima instrucción del microprograma, dado que la misma se recalcula en cada ciclo de reloj, lo que significa que no debe almacenarse para futuros ciclos.
Cuando la **microarquitectura** inicia la operación, un ciclo de inicialización coloca la micropalabra de la dirección 0 a la memoria de control en el registro de instrucciones de microprograma, para su ejecución. A partir de ese punto, se seleccionan las micropalabras a ejecutar desde alguna de las entradas **NEXT, DECODE o JUMP** del multiplexor de direcciones de la memoria de control, sobre la base de los valores que adoptan el campo **COND** del registro MIR de instrucciones de microprograma y la salida de la lógica de saltos de control. Luego de colocar cada palabra en el registro MIR, el trayecto de datos realiza las operaciones requeridas por los valores que adopten los diferentes campos del mismo registro.

Cada palabra de 41 bits comprende 11 campos distintos. A partir de la izquierda, el campo A determina cuál es el registro de trayectos de datos cuyo contenido debe colocarse sobre el bus A. Los direccionamientos binarios de registros se corresponden con la representación binarias. El campo AMUX selecciona si el decodificador A obtiene si el decodificador A obtiene su entrada desde el campo A del registro MIR (AMUX = 0) o desde el campo rs1 del registro %ir (AMUX = 1). Al igual B desde el campo rs2 de %ir.
El campo C determina en cuál de los registros del trayecto de datos de almacenará el dato transferido a través del bus C.

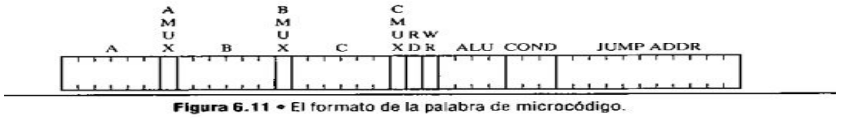


Figura 6.11 • El formato de la palabra de microcódigo.

Las líneas RD y WR determinan si se debe leer o escribir en memoria. Los campos no pueden estar simultáneamente en 1 pero si en 0. Para ambas operaciones la dirección de memoria se toma directamente del bus A. El dato que se ingresa a memoria se toma directamente desde el bus B y la salida de datos desde la memoria se coloca en el bus C. La línea RD controla el multiplexor de 64 a 32 bits del del bus C, el que determina si el bus C se carga desde memoria (RD = 1) o desde la unidad aritmético-lógica (RD = 0).
El campo **ALU** determina cuál de las operaciones aritmético-lógicas se ejecuta de acuerdo con los valores establecidos. El campo ALU puede adoptar 16 valores diferentes, no hay forma de apagar la unidad cuando no se la necesita, en esos casos debe elegirse una operación que no represente efectos colaterales indeseados.
El campo **COND (**salto condicional) de formato de la microinstrucción hace que el **microcontrolador** rescate la micropalabra siguiente, ya sea desde la posición siguiente en la memoria de control, o desde la posición indicada en el campo JUMP ADDR del registro MIR, o bien desde los bits del código de operación de almacenamiento %ir. Si el campo COND vale 000, no hay salto alguno, y se utiliza la entrada Next del multiplexor de direcciones de la memoria de control. La entrada **NEXT** se transfiere al circuito destinado al **incrementar las direcciones de la memoria de contro**l (CSAI,control store address incrementer), el que incrementa en 1 la salida actual del multiplexor de direcciones. Si el campo COND vale **001, 010, 100 o 101, se procede a realizar un salto condicional a la posición de la memoria de control que indica en el campo JUMP ADDR**, de acuerdo con las banderas n, z, v o c, o del bit 13 de %ir, respectivamente. La sintaxis “IR[13]” representa “el bit 13 del registro de instrucciones $ir” Si el campo COND vale 110, se produce un salto incondicional.
Cuando el campo COND vale 111, la dirección de memoria de control que debe copiarse en el registro MIR no se toma ni desde la entrada Next del multiplexor de direcciones ni de la entrada Jump, si no desde una combinación de 11 bits creada mediante el agregado de un 1 a la izquierda de los 30 y 31 bits de %ir y del agregado de 0 a la derecha de los bits 19-24 del mismo registro %ir .
El objetivo de usar este esquema de direcciones es el de permitir que la instrucción sea decodificada en un solo paso, a través de saltos a diferentes ubicaciones definidas por los valores que adoptan los campos op, op2 y op3 de la instrucción.

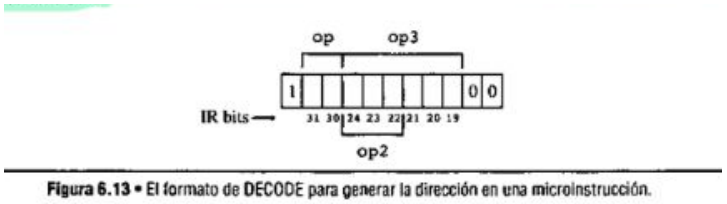


Figura 6.13 • El formato de DECODE para generar la dirección en una microinstrucción.

Finalmente en los campos JUMP ADDR aparece en los 11 bits menos significativos del formato de las micropalabras. Existen 2^11 micropalabras en l memoria de control por lo que se necesitan 11 bits de direccionamiento para poder acceder a cualquier posición de la memoria de control.

TEMPORIZADOR
La microarquitectura opera sobre un ciclo de reloj de dos fases, en el que las secciones maestras de todos los registros cambian con un flanco positivo de reloj, en tanto que las secciones esclavas de los registros cambian con un flanco negativo del mismo. Todos los registros usan biestables del tipo D de estructura maestro-esclavo activados por flancos negativos, excepto el %r0, el que no requiere biestable. En el flanco negativo del reloj, la información almacenada en la sección maestra de cada flip flop se transfiere hacia la sección esclava del mismo. Esta operación es la que deja disponible para las operaciones que involucran a la ALU. Cuando la función está en su estado bajo se

llevan a cabo las funciones ALU, MUX, CBL, las que deben haberse completado al momento del siguiente flanco de reloj. Los registros adoptan sus valores cuando la señal de reloj se encuentra en su nivel alto.

El desarrollo del microprograma

En una arquitectura microprogramada, las instrucciones se interpretan desde el microprograma almacenado en la memoria de control. Suele hablarse de **firmaware** cuando se hace mención al microprograma debido a que el mismo establece un puente entre el hardware y el software de la máquina.

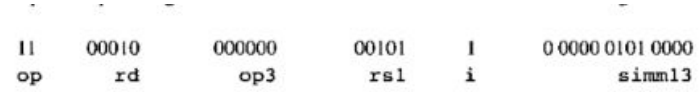
Ejemplo que muestra una porción de un microprograma que implementa el ciclo de búsqueda y ejecución de ARC (Leer Murdoca!)
En la memoria de control, cada microsentencia se almacena en forma codificada (ceros y unos) en una única micropalabra.
… Lenguaje microensamblador...
El lenguaje ensamblador de ARC es visible al usuario y se utiliza para la codificación de programas genéricos. Este lenguaje microensamblador se utiliza para la decodificación del firmware y no es accesible al usuario. El único propósito del firmware en el conjunto de instrucciones visible para el usuario. Una modificación en el conjunto de instrucciones del procesador involucra cambios en el firmware, en cambio una modificación a nivel del software escrito por el usuario no influye sobre el firmware.

Antes de iniciada la ejecución por parte del microprograma, el contador de programa se inicializó con la dirección de comienzo del programa cargado en la memoria principal.
La primera tarea en la ejecución de un programa nivel usuario es la de **cargar la instrucción** a la que apunta el contador de programa, **desde la memoria principal hacia el registro de instrucciones IR.**
Las líneas de dirección de memoria se toman desde el bus A. En la línea 0 del microprograma se carga el contador de programa en el bus A, se genera una operación de lectura de memoria.
La notación R[X] representa el registro x, tal que R[1] representa %r1.
La expresión AND(R[pc], R[pc]) realiza un producto lógico del contador de programa consigo mismo. Con el objetivo de **colocar %pc en el bus A,** hace falta elegir una operación de ALU que utilice el bus pero que no afecte los códigos de condición. Debe notarse que el resultado del producto lógico se DESCARTA debido a que el multiplexor del bus C de la figura 6.10 solo permite que la información que sale de memoria principal pase al bus C durante una operación de lectura.
Una **operación de lectura** requiere normalmente más tiempo para completar su ejecución que el tiempo requerido para la ejecución de una microinstrucción. El tiempo de acceso a la memoria principal varía según la organización de la memoria.
Con el objetivo de considerar las variaciones en los tiempos de acceso de la memoria el circuito incrementa las direcciones de la memoria de control (CSAI) no lleva a cabo el incremento de la dirección hasta que no se haya recibido una señal de reconocimiento (ACK) que indique que la memoria ha completado su operación.
El flujo de control dentro del microprograma se transfiere a la sentencia cuyo número de identificación sea el siguiente en sentido creciente, salvo que se detecte una operación de GOTO o de DECODE. En tal caso, en el ciclo siguiente se carga el MIR con la micropalabra.

Ahora que la instrucción se encuentra en el registro de instrucción como resultado de la operación de lectura ejecutada en la línea 0, el paso siguiente es el de la decodificación de los campos correspondientes al código de la operación. Esta operación se realiza ejecutando un salto hacia el microcódigo, de 256 posibilidades diferentes, según lo indica la palabra DECODE en la línea 256 del microprograma.
Se construye la dirección de salto, luego para decodificar los campos del código de operación, la ejecución del microcódigo continua en función de cuál de las 15 instrucciones de ARC está interpretando.
En la práctica, existe una cantidad de direcciones de DECODE que no debería aparecer nunca. No hay instrucción aritmética que responda a un formato binario 111111 en el capo op3, pero en previsión de que se produzca se deberá colocar en la dirección de DECODE 110 1111 1100 = (1788)10 una rutina de microcódigo capaz de lidiar con la instrucción ilegal.
Las Instrucciones de formato SETHI/Branch y call no tienen campos de operandos op3. Los formatos SETHI/Branch solo poseen campos op y op2, en tanto que call solo ofrece un campo op. Con el objetivo de mantener un mecanismo simple de decodificación, pueden crearse entradas duplicadas en la memoria de control.

Considerando el formato SETHI, si se sigue la regla determinada para obtener la dirección de DECODE, esta direccion debera tener un 1 es su bit más significativo, seguido por 00 en el campo OP seguido a su vez por 100 que identifique a SETHI en las posiciones 19-21 de la palabra, seguidos por los bits de posiciones 22-24 del registro de instrucción, finalmente seguidos por 00. Se obtiene 100100xxx00, en la xxx puede adoptar cualquiera de los valores posibles dependiendo del campo imm22. Los tres bits xxx pueden adoptar 8 valores diferentes, por lo que será necesario duplicar los códigos de SETHI en las posiciones de memoria identificadas con…(enumerar combinaciones)
Las direcciones de DECODE en los formatos de BRANCH Y CALL se construyen en posiciones duplicadas en forma similar.
Si bien este método de decodificación es simple y rápido, se pierde gran cantidad de memoria de control. Como solución alternativa se puede modificar el decodificador de la memoria de control tal que los formatos de salto de SETHI apunten a la misma dirección.

Se analiza la instrucción LD. El microprograma comienza en la posición 0, aun cuando no se sabe cual es el codigo de operación al que apunta el contador del programa en la memoria principal. La línea 0 del microprograma comienza la operación de lectura, tal como lo indica la palabra clave READ, con lo que se produce la carga de una instrucción desde la dirección de memoria principal, a la que apunta el contador de programa, hacia el registro de instrucciones.
SE SUPONDRÁ que el registro de instrucciones contiene el formato 32 bits siguiente:
11 00010 000000 00101 1 0 0000 0101 0000
op rd op3 rs1 i simm13
SIGUE: ld %r5 + 80, %r2 → 111 0000 0000 = 1792



En este ejemplo i = 1 por lo que se transfiere el control de la palabra ubicada en 1794, si i = 0 seria a la siguiente palabra 1793.
La línea 1792 suma los registros indicados en los campos rs1 y rs2 de la instrucción, anticipándose a la forma no inmediata de la instrucción ld, lo que solo tiene sentido su i = 0.
Dado que este no es el caso, el resultado almacenado en %temp0 se descarta al momento de la transferencia de control a la línea 1794.
En la palabra 1794 del microcódigo, se rescata el campo simm13 (utilizando extensión de signos, según lo indica la operación SEXT13), el que en la palabra 1795, se suma con el registro indicado en el campo rs1. Se transfiere luego el control a la palabra 1793, en la que se produce la operación de lectura. Se avanza ahora a la palabra 12074, en la que se incrementa el contador del programa con anticipación de la lectura de la siguiente instrucción a ser rescatada desde la memoria principal.
Dado que las instrucciones ocupan 4 bytes y deben estar almacenadas en posiciones de memoria cuyo valor sea múltiplo de cuatro, el contador de programa se incremente en en 4 unidades. Se devuelve el control a la línea 0 del microcódigo, en el que se repite el proceso. Se requiere así un conjunto de 7 microinstrucciones con el objeto de interpretar la instrucción LD.

Las instrucciones restantes, excepto en el caso de las instrucciones de salto, se interpretan en forma similar a la interpretación que se acaba de realizar para la instrucción ld. En las instrucciones de salto se requiere alguna modificación adicional debido a que el tipo de salto queda determinado por el campo COND del formato de las instrucciones de salto (25-28), campo que no se utiliza durante la operación de decodificación. La solucion que aqui se utiliza consiste en desplazar los bits del campo COND hacia IR[13] de a uno por vez, saltando luego a distintas locaciones del microcódigo según el formato binario de COND.
Para las instrucciones de salto, la operación DECODE de la línea 1 del microprograma transfiere el control a la línea 1088. Estas instrucciones de salto requieren más espacio que las cuatro palabras admitidas para cada instrucción, por lo que desde la línea 1088 se devuelve el control a la línea 2, en la que se inicia una sección suficientemente grande de memoria disponible para almacenamiento de control.

Las líneas 2-4 rescatan los 22 bits que corresponden al desplazamiento se saltó, colocan en ceros los 10 bits más significativos y almacenan el valor obtenido en el registro %temp0. Esto se logra corriendo %ir en 10 posiciones a la izquierda, almacenando en %temp0, y por ultimo corriendo el resultado nuevamente en 10 bits, ahora a la derecha.
Las líneas 5-7 desplazan $ir en 10 15 bits a la derecha de modo de almacenar el bit más significativo del campo COND (IR[28]) con la posición(IR[13]), lo que permite que una operación JUMP on IR[13] = 1 analice el valor de cada

bit. EN forma alternativa se podría desplazar el capo COND de IR[31] de a un bit por vez y utilizar la condición JUMP on n para verificar cada bit
El proceso de decodificación del salto se inicia en la línea 8. Si IR[28] que se encuentra ahora en IR[13] vale 1 la instrucción es BA, la que se ejecuta en la línea 12. Nótese que el control vuelve a la línea 0, en lugar de saltar a la línea 2047, para evitar el doble incremento del contador de programa en la misma instrucción.
Si IR[28] = 0, se desplaza %ir un bit a la izquierda a través de la suma del registro consigo mismo, de modo de alinear IR[27] en la posición IR[13]. El bit IR[27] se verifica en la línea 9. Si su valor es cero, se ejecuta la instrucción BE en la línea 10; en caso contrario, se desplaza %ir a la izquierda, verificando el estado de bit IR[28] en la línea 13. Las instrucciones de salto restantes se interpretan de forma similar.

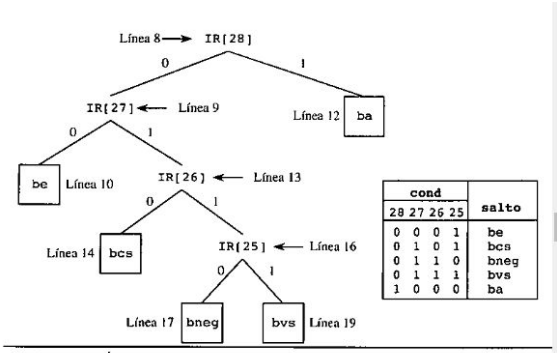


Figura 6.16 • Árbol de decodificación para las instrucciones de salto y sus correspondientes líneas de microprograma.

Traducción de lenguaje microensamblador

Un microprograma escrito en lenguaje microensamblador debe traducirse al código objeto binario antes de ser almacenado en la memoria de control.
Tal como debe traducirse un programa escrito en lenguaje simbólico antes de almacenarlo en memoria principal en forma de un programa objeto binario.

Cada línea del microprograma ARC corresponde a una palabra de la memoria de control. El microprograma se puede ensamblar linea por linea en un paso

0: R [ir] ← AND (R [pc], R [pc]); READ;
100 000 | 0 | 100 000 | 0 | 100 101 | 0 | 1 | 0 | 0101 | 000 | 0000 0000 000

El producto lógico del contador del programa consigo mismo inicia cargando %pc en los buses A Y B, se transfiere una palabra a través de la ALU. MUX = 0 -> la entrada de los multiplexores se toman desde el registro MIR.

1: DECODE
000 000 | 0 | 000 000 | 0 | 000 000 | 0 | 0 | 0 | 0101 | 111 | 0000 0000 000

EJEMPLO
si se requiere agregar otra operación al conjunto de instrucciones ARC. Se requiere modificar el microprograma para agregar la nueva instrucción.
Se inicia el proceso determinando la dirección de memoria de comienzo de la OPERACIÓN.

TRAPS E INTERRUPCIONES
Se define trap como el procedimiento automático de llamada generado por el hardware como consecuencia de una condición excepcional que se produce durante la ejecución de un programa. Ej instrucción ilegal, desborde, división x cero.
Cuando se produce un trap, se transfiere el control a un administrador de traps, rutina que es parte del sistema operativo.

Normalmente, existe una selección fija de memoria destinada a las direcciones de comienzo de los administradores de trap, en las que se almacena una única palabra para cada rutina de admisión. Esta selección de la memoria

conforman una Tabla de Saltos que permite que la dirección absoluta de cada situación pueda incluirse en el microcódigo, los destinos los puede modificar el usuario.
Un trap habitual es el que tiene que ver con las instrucciones de punto flotante, las que pueden ser emuladas por el sistema operativo. Si el microcódigo no conoce los códigos del punto flotante (implementadas por el hardware) se genera un trap y se transfiere el control al administrador de instrucciones legales y transfiere la rutina al emulador de punto flotante.
Actualmente las unidades de cálculo en formato punto flotante viene incorporadas dentro de los procesadores integrados.

Las interrupciones se producen luego de algún evento circuital considerado como una excepción.
Los traps son de naturaleza sincrónica con la ejecución de un programa en tanto que las interrupciones son asincrónicas. Un trap se producirá siempre en un mismo punto de un mismo programa que se ejecuta con el mismo conjunto de datos, mientras que las interrupciones son impredecibles.

Cuando se inicia la ejecución de alguna rutina de atención de una interrupción o de un trap, la misma procede a rescatar en la pila aquellos registros que piensa modificar, realiza su tareo, recupera los registros previamente almacenados y, luego, regresa al programa interrumpido. El proceso de retorno de un trap difiere del llamado de una subrutina debido a que también debe salvarse y luego rescatarse del registro %PSR. En ARC se utiliza la instrucción RETT. Puede requerirse una rutina de detección de interrupciones que eleve su prioridad (modo supervisor)

NANO PROGRAMACIÓN
Si la memoria de control es ancha y se repiten palabras puede ahorrarse espacio en memoria de microprograma colocando una copia de cada palabra de microcódigo en un elemento de NANOALMACENAMIENTO usando la memoria de microprograma como índice a la memoria de nanocodigo.
2048 palabras x 41 bits = 83.968 bits
Las micropralabras únicas forman un nanoprograma, el que se almacena en una memoria de lectura de 41 bits cada una.
El microprograma accede de forma indexada al nanocodigo. El microprograma tiene la misma cantidad de palabras independientemente de si una utiliza nanocodigo o no. Cuando se utiliza nanocodigo, en la memoria de control se almacena punteros.
Ahorra memoria, mejora eficiencia. Primero accede al microcódigo y luego al nanocodigo, funcionará más lentamente

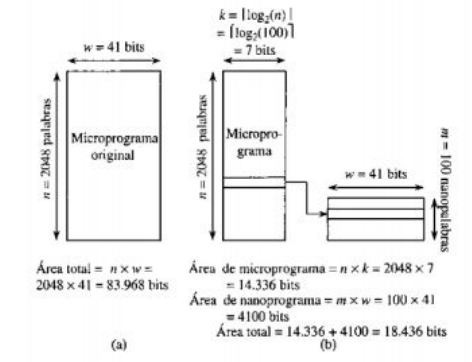


Figura 5.19 • (a) Microprogramación versus (b) nanoprogramación.

CONTROL CABLEADO
Una alternativa de la unidad de control microprogramada es el uso de un diseño cableado, en el que se realiza una implementación directa utilizando flip flops y compuertas lógicas, en lugar de usar elemento de almacenamiento y un mecanismo de selección de micro palabras. Los pasos de un microprograma se reemplazan por los estados de una máquina de estados finitos

CAPÍTULO 7 - Memoria

La jerarquía de memoria

Los registros son los más rápidos, de velocidad similar a la unidad de proceso, pero grandes y consumidor de mucha energía de alimentación.
En el fondo las memorias secundarias y los elementos de almacenaje off line, tal como los discos magnéticos y las cintas magnéticas, en los que el costo por bit almacenado es bajo en términos monetarios y de energía consumida, pero cuyo tiempo de acceso es muy alto comparado con el de los registros. Entre los registros y elementos secundarios se ubican otros tipos de memorias que tienen a salvar la brecha entre ambos extremos

Memoria de acceso aleatorio

La memoria de acceso aleatorio (RAM, random access memory) Puede accederse a cualquier celda de memoria en el mismo tiempo, independientemente de su posición.
Elemento de memoria usa ff tipo D, con entrada de lectura o escritura. Existe una línea de datos (bidireccional) para la entrada y salida de datos

Circuitos de memoria de acceso aleatorio basado en FLIP FLOP son de MEMORIA ESTÁTICA (SRAM, static RAM) debido a que cada posición de memoria se mantiene en tanto se mantenga la alimentación eléctrica del circuito integrado.
Los circuitos integrados de memoria dinámica, llamados DRAM, utilizan un capacitor que almacena una pequeña cantidad de carga eléctrica y el nivel de carga representa un 0 o 1. El capacitor es mas chico por lo que las memorias dinámicas almacenan más información en el mismo espacio que ocuparía una memoria estática. Las cargas del capacitor se van dispersando con el tiempo, la carga tiene que ser restablecida, refrescada con frecuencia

CIRCUITO INTEGRADO

En los terminales a0-am-1 se aplica una palabra de direcciones m bits, formada por m líneas numeradas de 0 hasta m-1. Se activa la señal CS (CHIP SELECT) junto con WR (escritura o lectura)
Cuando se lea información la palabra de datos aparecerá en las líneas D0-Dm-1 luego de un periodo de tiempo T (Retardo). Cuando se escriben los datos en un circuito integrado la línea de datos tiene que mantenerse estables por un periodo T
Las líneas de dirección A se decodifican a partir de una dirección de m bits a una dirección de 2^m bits, cada una de las cuales asocia una palabra de w bits. EL circuito integrado contiene, 2^m x w bits

Una memoria RAM puede considerarse como una selección de registros y un mecanismo de direccionamiento para permitir la selección de una palabra para su lectura o escritura. Líneas A seleccionar la palabra a ser leída a través del decodificador. La salida de los registros se interconectan y solo va a estar conectado un registro por vez, lo demás se desconecta eléctricamente por el medio de uso de buffers de 3 estados. CP no hace falta en la decodificación pero será necesaria para memorias más grandes.
En los circuitos integrados pequeños usan un solo decodificador para seleccionar una de 2m palabras. pero los
Un circuito integrado de 64M x 1 requiere 26 líneas de direccionamiento (64M == 2^26) lo que significa con decodificador de 2^26 compuertas y de 26 entradas x cada 1 → costo elevado en superficie para decodificar
Circuitos integrados son cuadrados y se accede por filas y columnas 2-1/2D. Deco de filas y columnas.
LECTURA: se selecciona una columna íntegra, la que ingresa al multiplexor de columnas, lo cual selecciona un único bit que será enviado al exterior de la memoria.
ESCRITURA: el demultiplexor de columnas orienta al único bit a ser escrit hacia la correspondiente columna, mientras que el deco de filas define en la que deberá escribirse dicho bit.
En la práctica para reducir la cantidad de terminales, el circuito integrado presenta solo m/2 terminales de direccionamiento. En esta caso se ingresa primero la dirección de la fila de m/2 bits, junto con la señal de sincronismo de filas RAS (row address strobe). Las direcciones de filas se almacenan en el circuito integrado y se decodifica. Igual columnas.
Se puede reducir todavía más los requerimientos de entradas y salidas con Árboles decodificadores.
Las RAM dinámicas son muy económicas, las memorias estáticas ofrecen mayor velocidad, Los ciclos de refresco, los circuito de detección de errores generan una diferencia de ¼ d velocidad
Normalmente suelen acceder a un conjunto de palabras que constituyen un bloque. Los accesos de memoria se pueden entrelazar de modo que mientras que una memoria accede a la dirección Am otras acceden a Am-1…

Memoria de Lectura
Cuando un programa se carga en la memoria se mantiene en memoria hasta que se lo sobreescriba o hasta que se apague la energía eléctrica. Para algunas aplicaciones el programa nunca cambia por lo que se puede fijar en una memoria de lectura ROM. Estas solo necesitan un decodificador, algunas líneas de salida y unas pocas compuertas logicas. NO hay necesidad de flip flops ni capacitores

MEMORIA CACHE
Cuando se ejecuta un programa la mayor parte de las referencias de memoria se hacen respecto a una pequeña cantidad de direcciones. PRINCIPIO DE LOCALIDAD. Cuando un programa hace una referencia a una localidad de memoria, muy probablemente vaya a acceder de nuevo a ella en el corto plazo -> LOCALIDAD TEMPORAL. También existe la LOCALIDAD ESPACIAL, en la cual se plantea que tras una referencia a una posición de memoria dada, es que muy probable que se acceda a posiciones cercanas a ella. La localidad temporal se produce debido a que los programas, en general, consumen mucho tiempo en iteraciones o en actividades recursivas, por lo que recorren la misma seccion de codigo una enorme cantidad de veces. La localidad espacial es porque los datos se almacenan en zonas contiguas en general. El acceso a memoria suele ser lento en comparación con la velocidad de la unidad central de proceso. EL principio de Localidad mejora el rendimiento.
Se puede colocar entre la memoria principal y la unidad de central de procesos una memoria cache, pequeña pero rápida, con el objeto de almacenar contenidos de las direcciones a las que se accede con mayor frecuencia. Durante la ejecución de un programa se analiza primero el contenido de la memoria cache, y se accede a la palabra requerida si estuviese. Si la palabra no está en cache se genera una posicion vacia y se carga la palabra requerida en esa posición desde la memoria principal, y luego se accede a cache a la palabra. Si bien lleva más tiempo que el acceso a memoria principal, el rendimiento en general mejora cuando se logra que una proporción alta de accesos a la memoria se satisfaga desde la memoria cache.

La memoria cache es mas veloz que la memoria principal. Una memoria cache tiene menor cantidad de direcciones que la memoria principal y como resultado tiene un árbol de decodificación poco profundo, lo que reduce el tiempo de acceso. Además la memoria cache está ubicada más cerca de la CPU, lo que evita los retardos en las transferencias sobre un bus compartido.

Memoria de asignación asociativa
Los bloques de memoria cache o LÍNEAS DE CACHE, tiene tamaños que van desde 8 hasta 64 bytes. La informacion ingresa y egresa de la memoria cache a una línea por vez, utilizando las técnicas de entrelazado de memoria. Existen más bloques de memoria principal que de cache y cualquiera de los bloques de la memoria principal puede colocarse en cache. Para mantener una pista de cuál de los posibles bloques de memoria se encuentra en cada línea de cache, se tiene un campo ETIQUETA
El campo etiqueta se ubica en los bits más significativos (el ejemplo usa 27) de los 32 bits presentes en cache. La etiquetas se almacenan en una memoria especial, en la que se puede buscar en paralelo.

Cuando se carga por primera vez un programa en la memoria cache, se limpia la memoria cache y durante la ejecución se requiere un BIT DE VALIDEZ para indicar si la línea tiene un bloque que pertenece o no al programa. Existe un BIT DE SUCIEDAD que controla si se produce una modificación de algún bloque durante su almacenamiento en la memoria cache. Si una línea se modifica tiene que ser re escrita en la memoria principal antes de que sea utilizada nuevamente por otro bloque.
Una referencia de localización en cache puede dar ACIERTO O FALLA. Al comienzo bit validez en 0, la primera instrucción provoca una falla de la memoria cache. Se ubica en la memoria principal el bloque que provocó la falla y se lo coloca en cache.

En Asignación Asociativa Cada bloque de la memoria principal puede asignarse a cualquier línea, y se realiza por medio de la partición de la dirección de campos, uno para la etiqueta y otro para la palabra (byte) [--27--|--5--]
Cuando se hace referencia a una dirección de memoria principal, la electrónica de la memoria cache intercepta esa referencia y busca dentro de la memoria de etiquetas para ver si el bloque requerido se encuentra almacenado en cache.
Para cada una de las líneas, se verifica si su bit de validez es 1, si lo es se compara con el campo de etiquetas de la dirección buscada con el campo de etiqueta de línea. Todas las etiquetas se analizan en paralelo, usando una memoria asociativa.

Si alguna etiqueta de la memoria de etiquetas coincide con el campo de etiquetas de la referencia de memoria principal, se extrae la palabra de la línea desde la posición indicada por el campo palabra. Si la palabra a la que se hizo referencia no se encuentra en la caché, se deberá traer desde la memoria principal el bloque de memoria que contiene la palabra buscada, recuperando luego esa palabra desde la memoria cache. Se actualizan los campos de etiqueta, validez y suciedad y se retorna a la ejecución del programa.

Reemplazo de memoria asociativa.
Cuando se requiere almacenar un bloque, se debe identificar alguna línea disponible (valides = 0) Cuando los bits de todas las líneas de cache valen 1, debe liberarse alguna de las líneas activas para cargar un nuevo bloque.
Criterios para liberar espacio:

- elemento menos usado (LRU): se agrega a cada línea una identificación de tiempo, que se actualiza cada vez que se accede a la línea
- primero en entrar primero que sale (FIFO)
- menos frecuentemente usado (LFU): contador de frecuencias, se libera el menos utilizado
- Aleatorio: elige una línea al azar
- Reemplazo óptimo: se usa con propósitos comparativos

VENTAJAS Y DESVENTAJAS DE ASIGNACIÓN ASOCIATIVA DE MEMORIA CACHE
Ventaja: cualquier bloque de memoria puede colocarse dentro de cualquier línea de memoria cache.
Desventajas: hace falta mucha electrónica para la administración y el control de cache.
En paralelo-> asociativas o direccionables por contenido
Si se restringe la posición de memoria cache en en el que se puede ubicar cada uno de los elementos del bloque de memoria principal -> asignación directa

ASIGNACIÓN DIRECTA
Cada línea de memoria cache se corresponde con un conjunto explícito de bloques de memoria principal. A cada bloque de memoria principal se le asigna una única línea, pero cada línea puede recibir más de un bloque. La asignación se realiza a través de la partición de la dirección en campos asignados a la etiqueta, la línea y la palabra.

Etiqueta 13 // línea 14 // palabra 5

Cuando se hace referencia a una dirección de memoria principal, el campo línea identifica en cuál de las 2^14 líneas puede encontrarse el bloque. Si el bit de validez es 1 se compara el campo etiqueta de la dirección de referencia con el campo de etiquetas de línea. Si los campos de etiquetas coinciden, la palabra se toma desde la posición de la línea especificada en el campo palabra.
Si el bit de validez pero los campos de etiqueta no coinciden, se vuelve a escribir la línea a la memoria principal si el bit de suciedad vale 1 y se lee de nuevo el bloque desde la memoria principal hacia la línea de memoria cache. Para un programa que recién inicia el bit de validez valdrá 0 así que directamente se escribe en la línea. Se coloca el bit de validez de esa línea en 1 y se reinicia la ejecución del programa
Ventajas. simple de implementar. No hay necesidad de una búsqueda asociativa, dado que el campo de línea de la dirección de memoria principal presentada por la CPU, se utiliza para dirigir la comparación a la única línea en que se podría encontrar el bloque

Sigue en capítulo 7.6.3 del Murdoca...