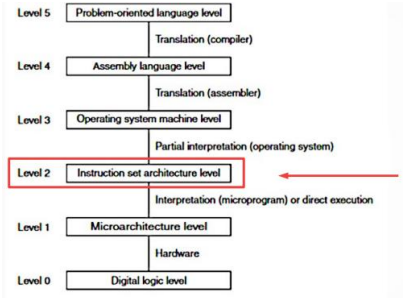
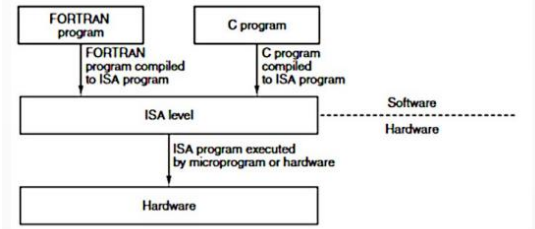
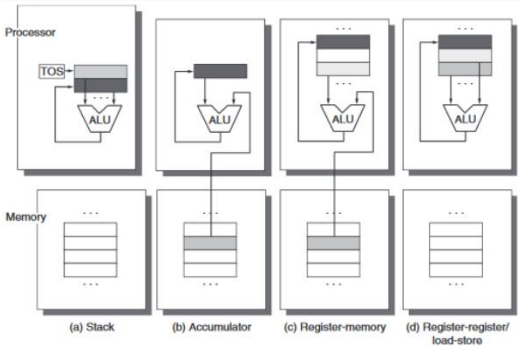
Nivel ISA (Instruction Set Architecture)
Es el nivel original de la maquina multinivel, y el más importante. Referido también como la arquitectura de la computadora. Es el nivel que integra el mundo de software con el mundo de hardware. Lenguaje que tanto los compiladores como el hardware tienen que entender para poder comunicarse uno con el otro
Un arquitecto de sistema tiene que: Negociar con los escritores de compiladores, con los ingenieros de hardware y tener presente compatibilidad en reversa.
Propiedades de un buen ISA: se debe definir un set de instrucciones que: Pueda ser implementadas en presente y futuras tecnologías, resultando en diseños que son eficientes en términos de costos, para variar generaciones de procesadores;Pueden proveer una traducción clara para el código compilado;Todo lo anterior manteniendo compatibilidad en reversa(realidad económica).
En muchas arquitecturas, el ISA es especificado por documentos formales, mantenidos por algún consorcio de la industria, lo que permite a los fabricantes poder crear sus propias implementaciones, con una variedad de precios y rendimiento, garantizando la funcionalidad a nivel software. El ISA es el código que el compilador entrega de salida.
Entonces, cuando hablamos de ISA, el escritor del compilador debe reconocer: Modelo de memoria;Qué registros existen y están disponibles;Qué tipos de datos están disponibles;Qué instrucciones están disponibles;*Qué hardware especializado tiene la micro-arquitectura*.
Taxonomía del nivel ISA: El factor primario que influye en el diseño de un ISA es el almacenamiento interno del procesador. Este tiene 3 opciones:Una pila;Un acumulador;Un set de registros(propósito general). Y la característica que afecta a este factor son los operandos de las instrucciones:En una arquitectura con pila, los operandos tienden a ser implícitos(existen en la pila);En una arquitectura con acumulador un operando es siempre el acumulador;En una arquitectura de registros de propósito general(GPR), los operandos son solo explícitos.
Arquitecturas Modernas: Las arquitecturas después de 1980 son principalmente arquitecturas GPR, y la mayoría usan Load-Store(solo se da el acceso a la memoria principal a cargar, moviendo un dato desde la memoria a un registro, o a guardar, moviendo datos de un registro a la memoria). Las razones de su aparición son: Almacenamiento interno al procesador(registros) es mucho mas rápido que la memoria principal;Los compiladores pueden usar a los registros de manera más eficiente que los otros tipos de almacenamiento interno(pila, acumulador);Los registros pueden ser usados para sostener variables.
Cuantos registros?Depende de la eficiencia del compilador. La mayoría reservan el uso de los registros para: Evaluar expresiones;Pasar parámetros;Y los que quedan para sostener variables.
Arquitecturas GPR: Estas mismas arquitecturas pueden dividirse en 3 subcategorias: Arquitecturas Registro-Memoria;Arquitecturas Registro-Registro(Load-Store);Arquitecturas Memoria-Memoria(no se encuentra hoy en día).
Características de arquitecturas GPR: Las instrucciones del ALU pueden tener dos operandos(la instrucción contiene dos operandos para la operación, y uno es re usado para guardar el resultado) o tres operandos(la instrucción tiene un operando para el resultado y dos operandos para la operación);La cantidad de operandos, en la instrucción del ALU, que pueden ser direccionados desde la memoria es de 0 a 3.
Ventajas y Desventajas de cada tipo de Arquitectura GPR: la primer numero es la cantidad de direcciones de memoria, y el segundo, el máximo numero de operandos permitidos
Registro-Registro(0,3): Ventajas: Simple, instrucciones de longitud fija, simple modelo de generación de código.Instrucciones toman un a cantidad similar de ciclos para ejecutar. **Desventajas:** Los programas se compilan a un total mayor de instrucciones que los mismos en arquitecturas que permiten referencias a memoria. Más instrucciones y una densidad menor de instrucciones que causan que el programa sea más grande.
Registro-Memoria(1,2): **Ventajas:** Los datos pueden ser accedidos sin tener que cargarlos. El formato de instrucción tiende a ser fácil de decodificar y suelen tener buena densidad. **Desventajas:** En una operación aritmética, un operando es destruido. Codificando el numero de registro y una dirección de memoria en cada instrucción puede restringir la cantidad de registros disponibles. CPI(Cicles-Per-Instruction) varia dependiendo donde está el operando.
Memoria-Memoria(2,2 o 3,3): Ventajas: Códigos más compactos. No desperdicia registros como almacenamiento temporario. **Desventajas:** Mucha variación del CPI, especialmente para instrucciones con tres operandos. Muchas variación en el trabajo por instrucción. Tantos accesos a memoria causa cuellos de botella.
Las 7 Dimensiones del ISA
Modelo de memoria: Se define como será interpretada y especificada una dirección de memoria.
Interpretación de Direcciones: Todas las memorias principales están divididas en celdas con direcciones consecutivas y hoy en día el tamaño de celda más común es la de 8bits(1byte). Pero también es común que el procesador utilice los datos en palabras de 4bytes o 8bytes. Todas ISAs recientes tienen varias instrucciones que proveen acceso a memoria a través de byte(8bits), half word(16bits), word(32bits), double word(64bits). Existen dos convenciones: **Big-Endian**, como “leer normal un numero”, donde el primer byte almacenado es el byte más significativo(0x12345678->Byte1=12,...,Byte4=78);**Little-Endian**, leer al revés, donde el primer byte almacenado es el menos significativo(0x12345678->Byte1=78,...,Byte4=12).
Luego, hablamos del alineamiento de la memoria, en la que se define si se permite o no que los objetos más grandes a 1byte se alineen o no, debido a una restricción de hardware resultando en menos lógica, circuitos mas simples y baratos. Se puede optar entonces dos opciones: no permitir referenciar una palabra de forma desalineada o poder hacerlo, pero, provocando que un acceso a una palabra desalineada deba ser resuelta por varias referencias alineadas(Flexibilidad a un costo alto de rendimiento, porque se hacen muchas llamadas de E/S).
Hay dos formas de tratar el espacio de direcciones: tratar la memoria como un solo espacio desde el punto de vista del nivel ISA, extendiendo desde 0 a 2^{32} - 1(para 32 bits) o 0 a 2^{64} - 1(para 64 bits) direcciones posibles. O la otra manera es donde se divide en dos espacios de direcciones, uno para instrucciones y otro para datos, lo que nos permite direccionar tanto 2^{32} bytes de instrucciones como 2^{64} bytes de datos con una interfaz de direccionamiento de 32bits; y ademas evitar sobrescribir las instrucciones de un programa ya que las escrituras solo van a espacio de datos.
Modos de direccionamiento: Las instrucciones de un ISA pueden tener operandos. Los operandos son efectivamente direcciones que apuntan adonde está el dato. La ISA debe especificar de que forma acceder al objeto en cuestión(indicado por el operando). Estas formas de acceder al objeto se denomina modos de direccionamiento. El modo de direccionamiento se extiende un poco más allá que del modelo de memoria, porque el operando puede indicar una dirección de memoria, un registro o un constante. Cuando se direcciona un lugar en memoria, la dirección específica en memoria es llamada la Dirección Efectiva.
Registro(Reg-Reg): Regs[R4] <- Regs[R4] + Regs[R3]. Este modo usa una dirección para seleccionar los registros correctos, ya que son un componente rápido y con direcciones cortas haciéndolo el método más utilizado. En arquitecturas modernas load/store, casi todas las instrucciones utilizan solo este modo y las únicas que usan algún modo de direccionamiento de acceso a memoria son las que leen un valor de memoria y lo colocan en un registro(load), y las que escriben un valor de un registro a memoria(store). Se busca maximizar la utilización de este modo en los compiladores, haciendo que las variables frecuentemente necesitadas sean cargadas en registros(como las variables que controlan un bucle).
Inmediato: Regs[R4] <- Regs[R4] + 3. Utiliza la manera más simple de direccionar un operando ya que la dirección del mismo es el valor. Este operando se conoce como **operando inmediato** porque la instrucción no tiene que hacer una referencia para buscar el valor, está disponible de forma inmediata en el momento en que es buscada(fetched) de memoria. Es útil para definir constantes(que sean pequeños, de valores enteros generalmente), y como limitación solo se tiene la cantidad de bits del restante tamaño de la instrucción para representar ese constante.
Direct o Absolute: Regs[R1] <- Regs[R1] + Mem[1001]. En este modo un operando contiene la dirección efectiva de memoria. Muy útil para definir variables globales donde su dirección puede ser determinada en tiempo de compilación. Parecido a direccionamiento inmediato, tiene la limitación en la cantidad de bits disponibles, en este caso para referenciar una dirección de memoria.
Register Indirect: Regs[R4] <- Regs[R4] + Mem[Regs[R1]]. Aquí la dirección efectiva de memoria se encuentra en un registro. Este uso se asocia con el término de puntero. Como ventaja sobre el anterior, puede referenciar memoria sin el costo de tener una dirección completa incrustada en la instrucción. También cuenta con la ventaja de poder modificar esa dirección. Útil para trabajar con arreglos.
Displacement: Regs[R1] <- Regs[R4] + Mem[100 + Regs[R1]]. Útil para direccionar a memoria desde un punto de referencia. La dirección efectiva se calcula desde una dirección base presente en la instrucción y un registro que contiene un desplazamiento. Útil para hacer operaciones entre elementos de dos arreglos de la misma longitud. (Tanenbaum lo llama Indexed).
Indexed (Based-Indexed): Regs[R3] <- Regs[R3] + Mem[Regs[R1] + Regs[R2]]. Al igual que el anterior, la base es contenido en un registro. Útil para operaciones que involucran matrices
Stack: En este modo no hay operandos, son implícitos y sus direcciones siempre son dos primeros valores que salen de una pila estilo LIFO. El resultado es empujado de vuelta a la pila. (no usado en arquitecturas GPR).
Tipos y tamaños de datos(y operandos): Los datos deben representarse de forma precisa, y se lo especifica en el nivel ISA. Hay una enorme flexibilidad para representar datos a nivel software, incluso para aquellos que no son soportados por el hardware, pero a costo de rendimiento.
En el ISA hay dos categorías superiores de tipos de datos:
Datos Numéricos: Enteros(integers): 8,16(short),32(int), y 64(long) bits;Signados(Complemento a dos) o No-signados.**Fraccionados(floating-points):**32(float),64(double),128(quadruple)bits;Signados(IEE754).
Representación Decimal (BCD): Cada dígito decimal es codificado en 4bits(Packed) o 8bits; Cuando se necesita que el valor fraccionado y la aritmética entre ellos sea exacta.
Datos No-Numéricos: Caracteres: 7bits para la decodificación ASCII (char); 16bits para Unicode(wide-chear, multi-byte-char). **Cadenas de caracteres(string):** Contienen un símbolo especial para indicar el final de la cadena; La instrucción puede contener un campo (operando para indicar la longitud de la cadena).**Booleano:** 8, 32 o 64 bits; En algunos sistemas FALSE es representado por el 0, mientras que TRUE por cualquier otro valor;Hay una representación especial del booleano cuando se encuentra en un arreglo, conocido como el bit map.**Puntero:** Usado para representar una dirección de memoria. 32 o 64 bits dependiendo del ancho del bus de direccionamiento.
Tipos de Operaciones: Los tipos de operaciones mas comunes son: Aritméticas y lógicas, Transferencia de datos, Control(branch, jump), Sistema(system calls), Punto flotante, Decimal, Cadena, Gráficos. Sin embargo, los tipos de operaciones disponibles dependen de las necesidades del mercado de software.
Transferencia de datos: Cuando necesitamos mover un dato, por ejemplo un dato de X dirección en memoria a un registro, hacemos una copia idéntica del dato en memoria y la información original en memoria permanece. Las instrucciones de movimiento tienen que indicar la cantidad de datos a ser movidos, que se encuentra explícito en la instrucción o implícito y siempre el mismo. Podemos observar 4 escenarios para mover datos: Memoria a Registro (LOAD), Registro a Memoria (STORE), Registro a Registro (MOVE), Inmediato(caso especial de registro-registro).
Operaciones Diádicas: Combinación entre dos operandos para producir un resultado, como la suma, resta, multiplicación, división, and, or, load, store. Las operaciones AND y OR son importantes para Masking(and, cuando se quiere extraer una porción de bits de una palabra) y para Packing(or, cuando se quiere empaquetar una porción de bits de una palabra).
Operaciones Monádicas: Solo necesitan un operando para producir un resultado, como los desplazamientos lógicos y aritméticos(shift), desplazamientos circulares(rotation), jump, br, goto, inc, dec, neg, clr. A veces una operación diádica tiene un equivalente monádica por ser muy usada, como INC que incrementa el operando por 1.
Instrucciones de entrada/salida: 3 esquemas: **E/S programada con espera activa,** cuando el procesador contiene instrucciones que inician una comunicación con un dispositivo y quedan en un bucle esperando la respuesta (utilizados en sistemas embebidos de tiempo real); **E/S controlada por interrupciones,** las instrucciones inician el dispositivo de E/S y se solicita que genera una interrupción cuando este listo (costoso para el procesador si el dispositivo genera muchas interrupciones); **E/S DMA (acceso directo a memoria),** parecido al de espera activa, pero en vez de que el procesador se encargue del dispositivo, hay un controlador llamado DMA que adquiere esta responsabilidad. El controlador tiene 4 registros: El 1ro contiene la dirección de memoria a ser leída o escrita, el 2do contiene el valor de la cantidad de bytes o words que tienen que ser transferidos, el 3roespecifica el numero o la dirección del dispositivo, el 4to especifica si los datos serán leídos del dispositivos o escritos al mismo (no tomar en cuenta el orden de los registros).
Tipos de control de flujo: La mayoría de instrucciones no alteran el flujo del programa, por lo que la secuencia de ejecución es en el orden en que aparecen en memoria. SI un programa contiene ramificaciones, el flujo del programa puede ser modificado de forma dinámica (run-time), provocando que la secuencia de ejecución y el orden en que aparecen en memoria ya no sea el mismo.
Lo que puede modificar el control del flujo del programa: Ramificaciones condicionales (branches), ramificaciones incondicionales(jumps), llamadas a procedimientos(calls), retorno de procedimientos(returns), trampas(traps), interrupciones(interrupts).
Opciones de ramificación: Código de condición (CC, ARM, PowerPC), prueba bits especiales establecidos por las operaciones de la ALU, bajo control del programa. Como ventaja, la condición se establece automáticamente con otras operaciones, pero como desventaja CC es un estado adicional. Los códigos de condición restringen el orden de las instrucciones ya que pasan información de una instrucción a una branch. **Registro de condición(Condition Register, MIPS),** prueba un registro arbitrario con el resultado de una comparación, es simple pero consume un registro. **Comparar y Bifurcar(Compare and branch PA-RISC),** La comparación es parte de la bifurcación y a menudo, dicha comparación se limita a un subconjunto. Permite que sea una instrucción en vez de dos para una bifurcación, pero resulta en demasiado trabajo por instrucción para la ejecución segmentada.
A veces se dispone de un registro especial que contiene una serie de banderas o códigos de condición, se lo nombra como Flag Register, Program Status Word, Status Register, y Condition Code Register. Este registro contienen una serie de bits que indican una condición, la cual una instrucción puede testar. Estos bits, por lo general, son generados automáticamente por el procesador si se cumple la condición en cuestión. Generalmente la dirección destino de la instrucción siempre tiene que ser explícito, siendo excepción cuando es una instrucción de retorno, ya que es una forma más simple de especificar el destino haciendo un desplazamiento relativo al PC(o de la instrucción actual) y este modo de direccionamiento se denomina relativo al PC(PC relative). Tiene varias ventajas este modo de direccionamiento, el destino es cercano al lugar donde se produce la ramificacion, requiere menos bits para representar el destino y permite al código ejecutarse independientemente de donde se lo carga a memoria (independencia posicional). Para poder implementar esta técnica, se debe designar la dirección de forma dinámica ya que puede cambiar entre diferentes tiempos de ejecución, guardando esos valores en registros o pilas y usando cualquiera de los modos de direccionamiento anterior.
Opciones de invocacion de procedimientos: Las llamadas y retornos de procedimientos involucran una transferencia de control y el guardado de estados. La dirección de retorno es indiscutible. Y existen dos convenciones: Dentro del procedimiento que hace el llamado, **caller saving,** o dentro del procedimiento llamado**, callee saving.** Hay que diferenciar además la llamada a un procedimiento y a una corutina, donde al llamar al primero, la dirección de retorno es guardada y el llamado usa la instrucción de retorno para transferir de vuelta el control. Y para el segundo, cada procedimiento llama al otro y continua una instrucción adelante desde la ultima llamada. Usa instrucciones diferentes a las típicas CALL y RETURN.
Trampas(Traps): Una trampa es un procedimiento iniciado de forma automática según una condición causada por un programa. La condición que los activa son excepciones causadas por un programa y detectadas por el hardware(es mas eficiente implementarlas a nivel hardware). Algunas traps comunes: Overflow y underflow en punto-flotante, Overflow en enteros, Violación de protección, Código operación no definido, Overflow en la pila, Dispositivo de E/S no existente, División por cero, Traer una palabra de una dirección impar.
Interrupciones: No es causado por el programa en ejecución sino por otra cosa(como un dispositivo de E/S). En una trampa se transfiere el control al manejador de interrupciones, mientras que aquí retorna el control al programa interrumpido. Puede decirse que las trampas funcionan de forma síncrona con el programa y las interrupciones son consideradas asíncronas. Como en cualquier transferencia de control, se guardan los estados de los registros y si una interrupción restaura completamente los estados de los registros a los de antes de la interrupción, se considera transparente. Puede pasar que se quiera realizar una interrupción en medio de otra interrupción, por lo que se soluciona de dos maneras: Cuando haya una interrupción, deshabilitarlas asegurando que se den de forma secuencial, pero provocando problemas cuando los dispositivos no pueden tolerar mucha espera. Y en la situación en la que no los dispositivos E/S tengan tiempo crítico, se asigna una prioridad a c/u de los dispositivos y asignar una prioridad al CPU. Para el último esquema se necesita que las interrupciones sean transparentes
Formato de Instrucción: Las instrucciones de un ISA están formados por un código, llamado
el opcode o código de operación, y información adicional, llamada operando, que indica dónde se encuentra el dato que va a ser manipulado. El formato de instrucción indica de que forma la instrucción va a ser codificada, que luego sera decodificada por el procesador lo más rápido posible. Se debe decidir como se codifica los modos de direccionamiento, el cual depende del rango de modos de direccionamiento y el grado de independencia entre el mismo y el opcode. En computadoras antiguas poseían bits extras en cada operando que indicaba cual era el modo de direccionamiento del mismo, y las ISAs que se implementaban de esta manera se llamaban ortogonales. En arquitecturas load-store, un máximo de un operando puede direccionar memoria con solo uno o dos modos de direccionamiento. En este caso, se codifica el modo de direccionamiento como parte del opcode. Otra cosa que se define es si sera variable o flexible. En el primer caso, produce programas compactos y es útil cuando se definen muchos modos de direccionamiento y operandos. En el segundo caso, combina la operación y el modo de direccionamiento en el opcode, y se define el mismo tamaño para todas las instrucciones, por lo que termina siendo útil para pocos modos de direccionamiento y operaciones. Más fácil de decodificar.
Para la longitud de la instrucción, mejor si la instrucción es corta y si el procesador puede buscar múltiples instrucciones en un solo pedido. Por lo que se crea una limitación en la cantidad que se puede direccionar, buscando una cantidad suficiente de bits en el opcode para expresar todas las operaciones deseadas y un margen más para futuras expansiones. Siempre teniendo en cuenta la cantidad de registros y el modelo de memoria.
Por lo que el objetivo es balancear el impacto del tamaño del campo de registro y modo de direccionamiento en el tamaño de la instrucción y por consecuencia el programa. Y además, instrucciones con una longitud que no provoque limitaciones de hardware(longitud múltiplo de bytes o palabras).
PROCESADOR I7 (su ISA se llama IA-32): tiene 3 formas de operar. **Real mode**: funciona como un
8086, las funciones nuevas se desactivan. **Virtual 8086 mode**: el so controla la maquina entera. Se
puede correr programas de forma aislada y protegida. **Protected mode**: se comporta como una i7
y habilita todas sus características (3 niveles: level0:kernel, level1y2:device drivers, level3:user
mode). El i7 tiene un espacio de direcciones enorme: 16384 segmentos de 32 bits, y usa palabras
de 32 bits con Little endian. **Formato de instrucción**: irregular y complejo, hasta 6 campos variables,
de los cuales 5 opcionales. CAMPOS: Prefix|0-5bytes|, opcode|1-2bytes|, mode|0-1byte|(define
el modo de direccionamiento y se divide en 3: MOD: indica si la instr es R-R R-M o M-R. REG: 3 bits,
R/M: 3 bits), sib|0-1byte|(scale,index,base), displacement|0-4bytes|(direccional de memoria), immediate|0- 4bytes|(constante). **Tipos de datos**: int con signo(8,16,32,64bits), int sin signo (8,16,32,64b), BDC(8), Float(32,64).
OMAP4430: es un SOC (system on chip) donde el procesador, ARM Cortex A9, implementa el ISA ARM v7. A diferencia del i7, usa RISC, es decir, LOAD-STORE. Tiene 2 modos de operación: Kernel Mode/ User mode. A diferencia del IA-32, el **formato de instrucción del ARM v7** es limpio. Tiene instr de longitud fija, de 16 y 32 bits. **Modos de Direccionamiento**: Register, immediate, displacement, register indirect, pc relative. **Tipos de datos**: Enteros signados con complemento a 2 (8,16,32,64bits), entero sin signo(8,16,32,64), Float(32,64), NO tiene BCD. **Dato**: tiene mucho menos instrucciones que el IA-32 (el ISA del i7).

Modulo I

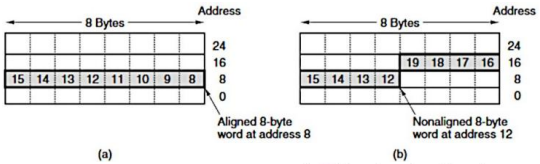


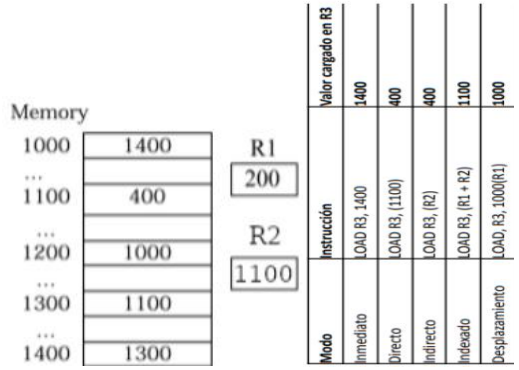
Estamos aquí





Modulo II





Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a r
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local varia (+ simulates register i direct addressing mo
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a po computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in a addressing: R1 = base R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for static data; address co need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of then mode yields *sp*.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + *d*	Useful for stepping th within a loop. R2 poir array; each reference R2 by size of an elem
Autodecrement	Add R1, –(R2)	Regs[R2] ← Regs[R2] – *d* Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincr Autodecrement/-incr also act as push/pop t a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * *d*]	Used to index arrays. applied to any indexe mode in some compu

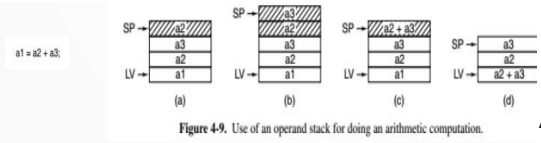
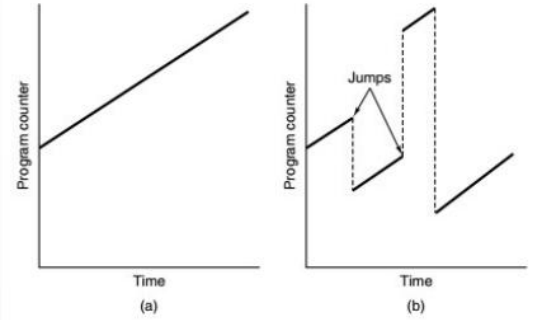
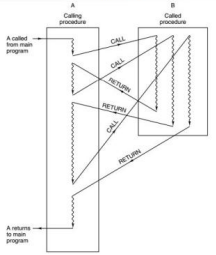


Figure 4-9. Use of an operand stack for doing an arithmetic computation.

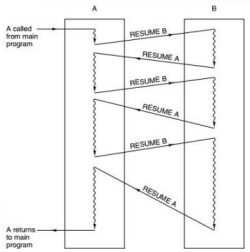
Modulo III

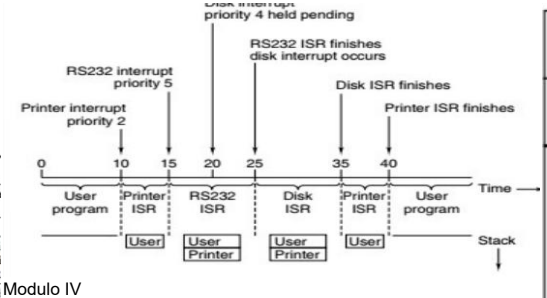


Llamado a procedimiento



Llamado a corutina





Modulo IV

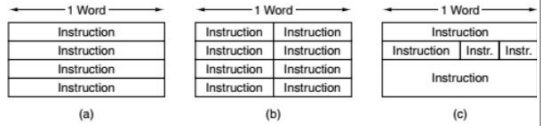
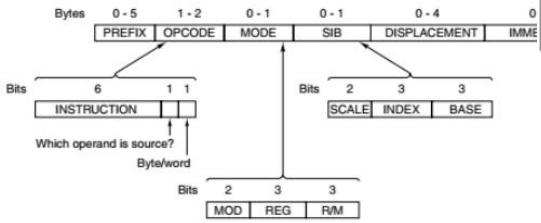


Figure 5-10. Some possible relationships between instruction and word length.

Procesador i7



ARM v7 OMAP4430