3.1.2 Repertorio de instrucciones y formato de la instrucción máquina

Parte III. Diseño del procesador
Módulo 3.1 Lenguaje máquina y ensamblador

Ingeniería Técnica en Informática

Facultad de Informática - Universidad Complutense de Madrid

Clasificación de las instrucciones (1/2)

- Transferencia de datos, permiten el movimiento entre distintos dispositivos de almacenamiento del computador
- Aritméticas, permiten realizar operaciones de tipo aritmético
- Lógicas y de manipulación de bits, permiten realizar operaciones lógicas, bit a bit, entre los operandos o manipular un bit del operando
- Desplazamiento y rotación, permiten desplazar o rotar un operando a la decha. o la izda. un nº determinado de bits
- Control de flujo, permiten romper la secuencia normal de ejecución

Clasificación de las instrucciones (2/2)

- Otras instrucciones
  - Transformación de datos
  - Manipulación de direcciones
  - Creación de marcos de almacenamiento local
  - Control del sistema
  - Entrada / Salida

Transferencia de datos (1/2)

- Permiten el movimiento entre distintos dispositivos de almacenamiento del computador (registros, memoria y pila)
- Necesario especificar
  - Tipo de movimiento
  - Dirección de operandos fuente y destino
  - Tamaño de datos a mover (byte, palabra, doble palabra, ...)
  - Nº de elementos a mover (para movimientos múltiples)

Transferencia de datos (2/2)

Instrucción	Operación	Descripción
`MOVE fnte, dest`	dest ← fnte	Transfiere palabra de reg. a reg., reg. a mem, mem. a reg o mem. a mem. (fuente= mem. o reg.; destino = mem. o reg.)
`LOAD Ri, dir`	Ri ← dir	Transfiere palabra de memoria a registro
`STORE dir, Ri`	Ri ← dir	Transfiere palabra de registro a memoria
`PUSH fnte`	Pila ← fnte	Transfiere palabra de mem. o reg. a la cabecera de pila
`POP dest`	dest ← Pila	Transfiere palabra de cabecera de pila a mem. o reg.
`MOVEM fnte, dest, n`	$dest_0 \leftarrow fnte_0$ $dest_{n-1} \leftarrow fnte_{n-1}$	Transfiere n palabras a partir de una dir. inicial fuente y una dir. inicial destino

Instrucciones aritméticas (1/2)

- Permiten realizar operaciones de tipo aritmético
- Necesario especificar
  - Tipo de operación (suma, resta, multiplicación, división, etc.)
  - Tipo de operandos y de aritmética (con signo, sin signo, entera, punto flotante, BCD, ...)
  - Tamaño de datos sobre los que se opera
  - Dirección de operandos fuente y destino (0, 1, 2 ó 3, según el repertorio)

Instrucciones aritméticas (2/2)

Instrucción	Operación	Descripción
`ADD fnt1,fnt2,dest`	dest←fnt1+fnt2	Suma dos operandos
`SUB fnt1,fnt2,dest`	dest←fnt1-fnt2	Resta dos operandos
`MULT fnt1,fnt2,dest`	dest←fnt1×fnt2	Multiplica dos operandos
`DIV fnt1,fnt2,dest`	dest←fnt1/fnt2	Divide dos operandos
`NEG fnte,dest`	dest←-fnte	Cambia de signo al operando
`ABS fnte,dest`	dest←abs(fnte)	Obtiene el valor absoluto
`INC fnte,dest`	dest←fnte+1	Suma 1 al operando
`DEC fnte,dest`	dest←fnte-1	Resta 1 al operando
`COMP fnt1,fnt2`	fnt1-fnt2	Resta y activa los bits de estado según resultado

Instrucciones lógicas y de manipulación de bits (1/2)

- Permiten realizar operaciones lógicas, bit a bit, entre los operandos o manipular un bit del operando
- Necesario especificar
  - Tipo de operación (AND, OR, NOT, Bit Clear, Bit Set, etc)
  - Tamaño de datos sobre los que se opera (byte, palabra, doble palabra, ...)
  - Dirección de operandos fuente y destino (0, 1, 2 ó 3, según el repertorio)
  - El número de bit en las instrucciones de manipulación de bit

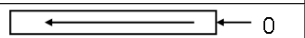
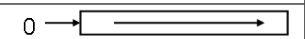
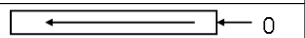
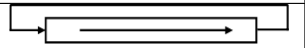
Instrucciones lógicas y de manipulación de bits (2/2)

Instrucción	Operación	Descripción
`AND fnt1,fnt2,dest`	dest←fnt1∧fnt2	Y lógica
`OR fnt1,fnt2,dest`	dest←fnt1∨fnt2	O lógica
`NOT fnte,dest`	dest← ¬fnte	Negación
`XOR fnt1,fnt2,dest`	dest←fnt1⊕fnt2	O exclusiva
`BCLR dest,n`	dest(n)←0	Pone a 0 el bit n
`BSET dest,n`	dest(n)←1	Pone a 1 el bit n
`BTEST fnte,n`	Estado←fnte(n)	Activa flag según bit n

Desplazamiento y rotación (1/2)

- Permiten desplazar o rotar un operando a la decha. o la izda. un nº determinado de bits
- Necesario especificar
  - Tipo de operación (desplazamiento izda. o decha., rotación izda. o decha.,)
  - Tamaño de datos sobre los que se opera (byte, palabra, doble palabra, ...)
  - Dirección del operando
  - Nº de bits a desplazar o rotar

Desplazamiento y rotación (2/2)

Instrucción	Operación	Descripción
`LSL fnte,n`		Desp. lógico izda. n bits
`LSR fnte,n`		Desp. lógico dcha. n bits
`ASL fnte,n`		Desp. aritm. izda. n bits
`ASR fnte,n`		Desp. aritm. dcha. n bits
`RL fnte,n`		Rotación izquierda n bits
`RR fnte,n`		Rotación derecha n bits

Control de flujo (1/4)

- Permiten romper la secuencia normal de ejecución y saltar a una determinada dirección especificada en la instrucción o implícita
- Necesario especificar
  - Dirección de siguiente instrucción a ejecutar si el salto es explícito
- Las instrucciones de bifurcación o salto condicional (Bcc), saltan o no en función de la condición especificada (cc)
- Esta condición se calcula a partir de los bits de condición del registro de estado
  - Los bits de condición se activan según el resultado de las instrucciones que se detallan en las siguientes tablas

Control de flujo (2/4)

Instrucción	Operación	Descripción
`JMP dir`	PC←dir	Salta (Jump) a dir
`Bcc dir`	if(cc) PC←dir else PC←PC+longInstr	Bifurca (Branch) a dir si (cc) es cierta
`JSR dir`	Pila←(PC,STR) PC←dir	Salto a subrutina Guarda PC y estado
`RTS`	(PC,STR)←Pila	Retorno subrutina, recupera PC y STR
`SKIP n`	PC←PC+n×longInstr	Salta n instrucciones
`NOP`	PC←PC+longInstr	No hace nada, siguiente instr.

Control de flujo (3/4)

cc	Nombre	Cierta si
`EQ`	Igual (Equal)	En una comparación previa los operandos son iguales
`NEQ`	No igual (No equal)	En una comparación previa los operandos son distintos
`GT`	Mayor que (Greater Than)	En una comparación previa el primer operando es mayor que el segundo
`GE`	Mayor o igual (Gr. or Eq.)	En una comparación previa el primer operando es mayor o igual que el segundo
`LT`	Menor que (Less Than)	En una comparación previa el primer operando es menor que el segundo
`LE`	Menor o igual (Les.or Eq.)	En una comparación previa el primer operando es menor o igual que el segundo
`Z`	Cero (Zero)	El resultado de una operación anterior es cero
`NZ`	No cero (No Zero)	El resultado de una operación anterior es distinto de cero

Control de flujo (4/4)

cc	Nombre	Cierta si
`P`	Positivo	El resultado de una operación anterior es positivo
`N`	Negativo	El resultado de una operación anterior es negativo
`C`	Acarreo (Carry)	El resultado de una operación anterior ha producido acareo
`NC`	No acarreo (No Carry)	El resultado de una operación anterior no ha producido acareo
`V`	Desbord. (Overflow)	El resultado de una operación anterior ha producido desbordamiento
`NV`	No desbord. (No Overfl.)	El resultado de una operación anterior no ha producido desbordamiento
`T`	Verdad (True)	Siempre cierta
`F`	Falso (False)	Siempre falsa

Otras instrucciones (1/2)

- Transformación de datos
  - Cambian el formato de los datos, por ej. de decimal a binario
- Manipulación de direcciones
  - Permiten calcular la dirección efectiva de un operando y almacenarla en un registro o en pila

```
LEA fnt,reg      (Load effective address: reg ← Dir de fnt)
PEA fnt          (Push Effective Address: Pila ← Dir de fnte)
```

- Creación de marcos de almacenamiento local
  - Permiten reservar espacio en la pila del sistema, p. ej:

```
LINK reg,#tam    (Pila←reg, reg←SP, SP←SP+tam)
UNLK reg         (SP←reg, reg←Pila)
```


Otras instrucciones (2/2)

- Control del sistema
  - Suelen ser instrucciones privilegiadas que usa el sistema operativo

```
RESET            (para reiniciar el computador: PC←valor inicial)
RTE              (retorno de excepción o interrupción)
```

- E/S
  - Para entrada y salida de datos entre el computador y dispositivos periféricos

```
IN dirPerif,reg   (reg←Periférico)
OUT dirPerif,reg  (Periférico←reg)
```


Ejecución alternativa

- Según el valor de cierta condición o valor, se decide entre varios caminos de ejecución
  - Simple: Un sólo `if`
  - Doble: `if..else`
  - Múltiple: Múltiples `if..elseif`
  - Selectiva: `switch, case`

Ejecución alternativa simple

```
...
if (a cc b) then
  bloque
end
...
```

```
    ...
    CMP a,b
    BNcc end
    bloque
end ...
```

Ejecución alternativa doble

```
...
if (a cc b) then
  blqthen
else
  blqelse
end
...
```

```
      ...
      CMP a,b
      BNcc else
      blqthen
      JMP end
else  blqelse
end   ...
```


Ejecución alternativa múltiple

```
...
if (a0 cc0 b0) then
  bloque0
elseif (a1 cc1 b1) then
  bloque1
...
elseif (an ccn bn) then
  bloquen
else
  blqelse
end
...
```

```
      CMP a0,b0
      BNcc0 if1
      bloque0
      JMP end
if1   CMP a1,b1
      BNcc1 if2
      bloque1
      JMP end
      ...
ifn   CMP an,bn
      BNccn else
      bloquen
      JMP end
else  blqelse
end   ...
```


Ejecución alternativa selectiva

```
...
case a of
  0: bloque0
  1: bloque1
  ...
  n: bloquen
end
...
```

```
tabla DATA etq0
      DATA etq1
      ...
      DATA etqn
      ...
      MOVE a,D1
      LEA tabla(D1*tam_dir),A1
      JMP (A1)
etq0  bloque0
      JMP end
etq1  bloque1
      JMP end
      ...
etqn  bloquen
end   ...
```


Ejecución iterativa

- Según el valor de cierta condición, se decide repetir la ejecución de cierto bloque de código
  - Con evaluación al final: `repeat..until`
  - Con evaluación al principio: `while`
  - Con un número definido de iteraciones: `for i=1,n`

Con evaluación al final

```
...
repeat
  bloque
until a cc b
...
```

```
        ...
repeat  bloque
        CMP a,b
        BNcc repeat
        ...
```


Con evaluación al principio

```
...
while a cc b do
  bloque
end
...
```

```
        ...
while   CMP a,b
        BNcc end
        bloque
        JMP while
end     ...
```


Con un número definido de iteraciones

```
...
for c := n to m do
  bloque
end
...
```

```
     ...
     MOVE n,c
for  CMP m,c
     BGT end
     bloque
     INC c
     JMP for
end  ...
```


Llamada a subrutina (1/5)

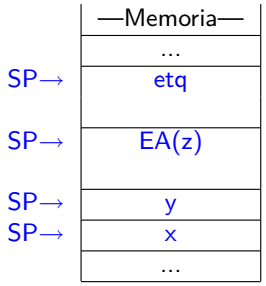
- Supongamos:
  - Una memoria direccionada por bytes
  - Un tamaño palabra de 16 bits
  - Un tamaño de dirección de 32 bits
  - que un integer se representa mediante palabra

```
...
procedure sub( a, b :integer; var c : integer )
begin
 bloque
end
...
sub(x,y,z);
...
```

Llamada a subrutina (2/5)

Parámetros

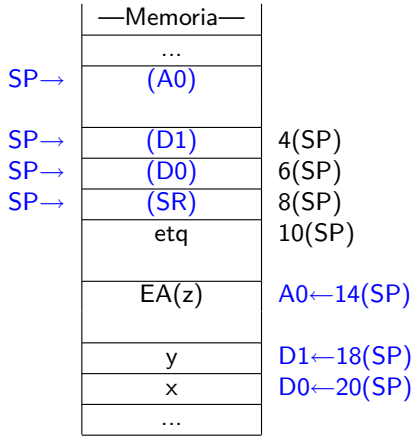
```
sub(x,y,z)
PUSH x
PUSH y
PEA z; !!
JSR sub
etq ...
```



Llamada a subrutina (3/5)

Acceso a parámetros

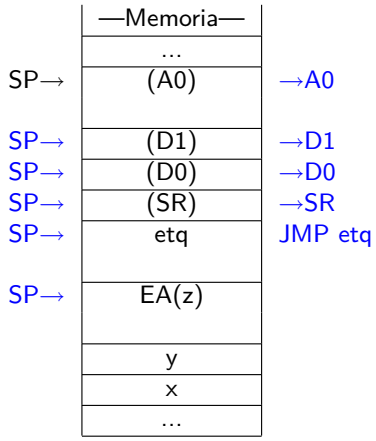
```
;Salva contexto
sub PUSH SR
PUSH D0
PUSH D1
PUSH A0
MOVE 20(SP),D0 ;a
MOVE 18(SP),D1 ;b
MOVE 14(SP),A0 ;c en
&z
bloque
```



Llamada a subrutina (4/5)

Restaura contexto

```
POP A0
POP D1
POP D0
POP SR
RTS
```



Llamada a subrutina (5/5)

```
; paso param
    PUSH x
    PUSH y
    PEA z
; llamada
    JSR sub
// liberación param
etq ADD #8,SP
    ...
```

```
sub PUSH SR ; salva contexto
    PUSH D0
    PUSH D1
    PUSH A0
; acceso a param
    MOVE 14(SP),D0
    MOVE 18(SP),D1
    MOVE 20(SP),A0
    bloque
; restaura contexto
    POP A0
    POP D1
    POP D0
    POP SR
    RTS ; retorno a etq
```

Subrutina con variables locales (1/5)

Llamada a subrutina: Variables locales

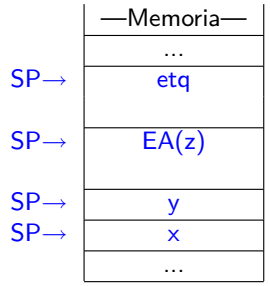
Realizar la traza anterior con el siguiente fragmento de programa

```
...
procedure sub(a,b:integer; var c:integer)
  var loc1, loc2, loc3 :integer
begin
  bloque
end
...
sub(x,y,z);
...
```


Subrutina con variables locales (2/5)

Parámetros

```
sub(x,y,z)
PUSH x
PUSH y
PEA z; !!
JSR sub
etq ...
```

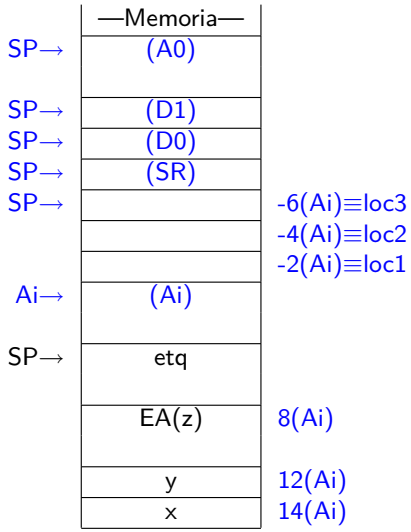


Subrutina con variables locales (3/5)

Acceso a parámetros

```
;Puntero de marco
sub LNK Ai,#-6
;Salva contexto
PUSH SR
PUSH D0
PUSH D1
PUSH A0
bloque
```

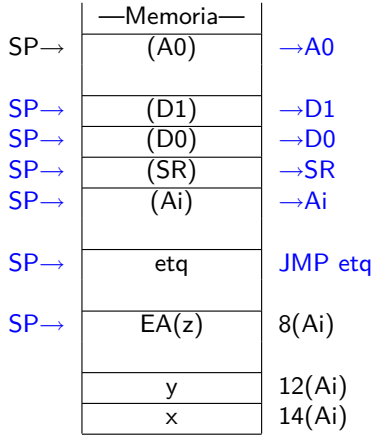
El mapa de memoria de la figura constituye el marco de activación de la subrutina.



Subrutina con variables locales (4/5)

Restaura contexto

```
POP A0
POP D1
POP D0
POP SR
UNLK Ai
RTS
```



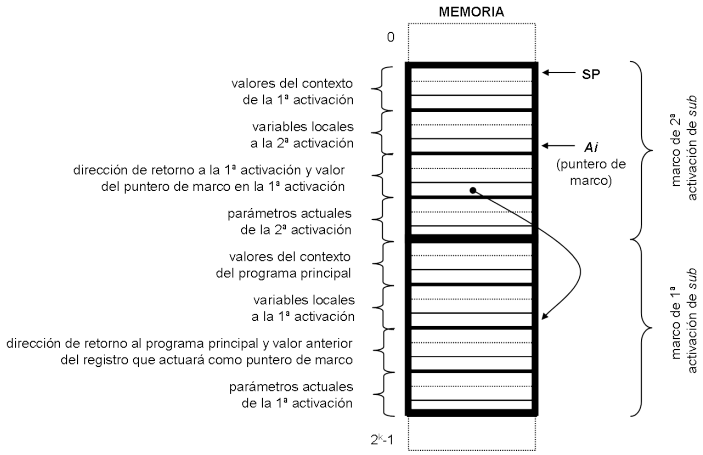
Subrutina con variables locales (5/5)

```
; paso param
    PUSH x
    PUSH y
    PEA z
; llamada
    JSR sub
// liberación param
etq ADD #8,SP
    ...
```

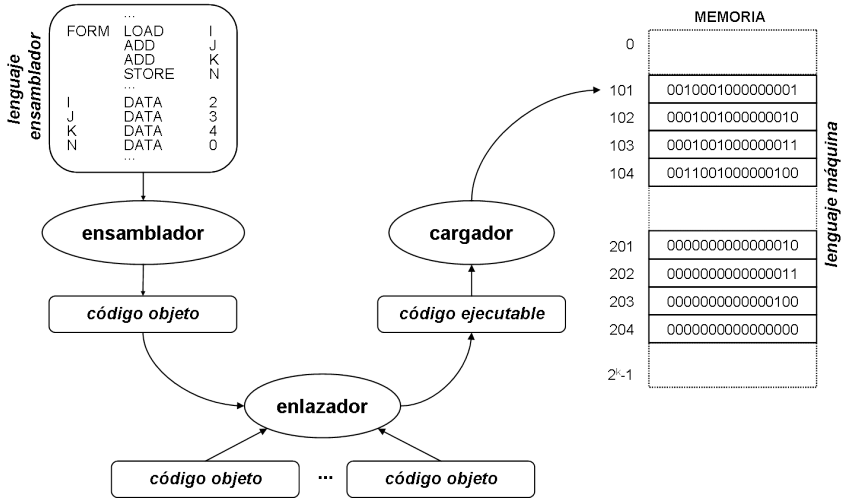
```
; área local de variables
sub LNK Ai,#-6
    PUSH SR ; salva contexto
    PUSH D0
    PUSH D1
    PUSH A0
    bloque
; restaura contexto
    POP A0
    POP D1
    POP D0
    POP SR
    UNLK Ai
    RTS ; retorno a etq
```


Anidamiento de subrutinas

- Suponiendo que la subrutina "'sub"' es recursiva, la evolución de la pila tras dos llamadas es la siguiente.



De código ensamblador a programa en memoria



De código ensamblador a programa en memoria
Ensamblador (assembler)

- Resuelve las etiquetas de instrucciones y datos
  - Relativas a PC para instrucciones
  - Relativas a algún registro para datos
- Expande macros y pseudo-instrucciones
- Interpreta las directivas de ensamblaje
- Fija la representación de los datos
- Traduce las instrucciones a código máquina
- Crea el fichero objeto
  - Determina cabeceras, segmentos de código y segmentos de datos
  - Crea tablas de símbolos: símbolos no resueltos + símbolos visibles
  - Añade información para la depuración

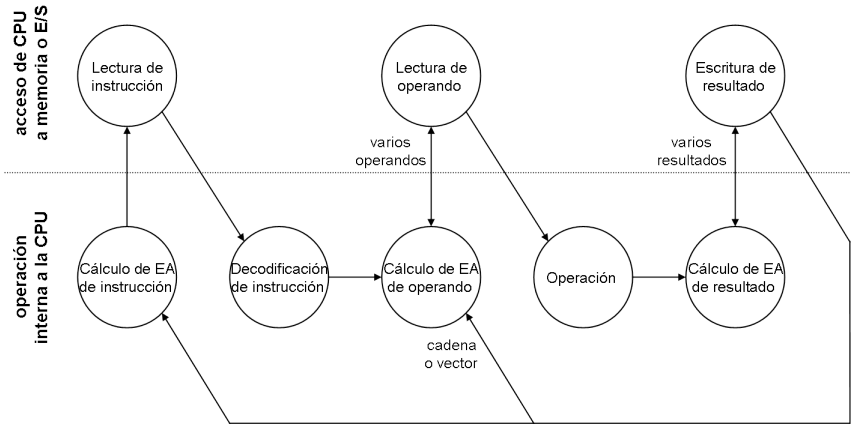
De código ensamblador a programa en memoria
Enlazador (Linker)

- Combina varios códigos objeto, resolviendo las referencias cruzadas
- Crea un código ejecutable

De código ensamblador a programa en memoria
Cargador (Loader)

- Lee y carga sobre memoria el código ejecutable
- Inicializa registros, pila y argumentos
- Fija los vectores de excepción
- Salta a la rutina de inicio del programa

De código ensamblador a programa en memoria
Ciclo instrucción



Elementos constitutivos de una instrucción máquina

- Código de operación (opcode)
  - Especifica el tipo de operación a realizar: Suma, Resta, Y lógica, Movimiento de datos, etc.
- Operandos fuente
  - La instrucción puede involucrar uno o más operandos fuente (o ninguno)
  - Estos podrán especificarse utilizando distintos modos de direccionamiento
- Operando destino
  - Si la instrucción produce un resultado deberá especificar el operando destino
  - Este podrá especificarse utilizando distintos modos de direccionamiento

Elementos constitutivos de una instrucción máquina

- Instrucción siguiente
  - En ocasiones es necesario especificar cuál es la siguiente instrucción a ejecutar
  - Normalmente la ejecución del programa es secuencial, donde la dirección de la siguiente instrucción suele ser implícita:
    - `Dir. instr. siguiente = Dir. instr. actual + Longitud instr. actual`
  - En las instrucciones de salto sí es necesario especificar la dir. de la siguiente instrucción
    - Ruptura de la secuencia normal del programa

Instrucción máquina

opcode	op. fuente 1	.....	op. fuente n	op. destino	instr. siguiente

Alternativas de diseño del formato de instrucción

- A la hora de diseñar el formato de las instruc. máquina de un repertorio, debemos decidir
  - Cuál será la longitud de las instrucciones
    - Todas las instrucciones de igual longitud
    - Instrucciones de distinta longitud, según el tipo de operación a realizar
  - De cuántos campos constará la misma (dependiendo del tipo de instrucción)
  - Cuántos bits ocuparán cada uno de esos campos
  - Codificación de cada campo

Alternativas de diseño del formato de instrucción

- Los principales factores a tener en cuenta para decidir este formato son
  - Número de operaciones distintas presentes en el repertorio
    - Aritméticas, lógicas, de control, de movimiento de datos, ...
  - Número de operandos de la instrucción (incluyendo operandos fuente y destino)
    - Instrucciones sin operandos
    - Instrucciones con 1, 2, 3, ... operandos
  - Modos de direccionamiento disponibles para cada operando
    - Inmediato, absoluto, de registro, indirecto, ...
  - Tamaño y tipos de datos soportados
    - Bit, byte, palabra, doble palabra, ...
    - Caracteres, BCD, magnitud y signo, complemento a 2, punto flotante, ...

Introducción

- Una operación típica de un computador (p. e. `A=B+C`) suele tener tres operandos
  - Dos operandos fuente `B` y `C` y un operando destino `A`
- Las principales alternativas que se presentan son
  - Instrucciones con tres operandos explícitos
  - Instrucciones con dos operandos explícitos
  - Instrucciones con un único operando explícito
  - Instrucciones sin operandos explícitos

Instrucciones con tres operandos explícitos

- Se especifican los tres operandos en la instrucción máquina
  - Cada operando puede soportar distintos modos de direccionamiento
- Formato flexible, pero ocupa mucho espacio

Ejemplo

`ADD A,B,C` (A←B+C)

Instrucciones con dos operandos explícitos

- Dos operandos explícitos y uno implícito
  - Uno de los operandos actúa como fuente y destino
  - Cada operando puede soportar distintos modos de direccionamiento
- Es menos flexible, pero ocupa menos espacio

Ejemplo

`ADD B,C` (B←B+C)

Instrucciones con un operando explícito

- Se especifica un sólo operando en la instrucción máquina
  - Uno de los operandos y el resultado se almacenan en un registro especial del procesador: el acumulador
  - El operando puede soportar distintos modos de direccionamiento
- Formato muy reducido, pero poco flexible

Ejemplo

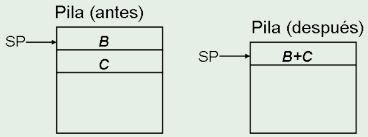
`ADD B` (Acum.←B+Acum.)

Instrucciones sin operandos explícitos

- Tanto los operandos como el resultado se almacenan en la pila del sistema
- Ocupa muy poco espacio
- Requiere varias instrucciones previas de acceso a la pila

Ejemplo

`ADD` ((SP)←(SP)+(SP+1))



Ejemplo: Número de operandos

Ejemplo

Escribir los programas simbólicos para la siguiente operación:
$Y = (A - B)/(C + D * E)$
en un computador de 0, 1, 2 ó 3 operandos, utilizando las instrucciones que sean necesarias y sin sobreescribir el contenido de ninguno de los operandos fuente.

Ejemplo: Diseño de instrucción máquina

Diseño de instrucción máquina

Diseñar el formato de las instrucciones de un computador con 8 registros de propósito general que permita codificar en una instrucción de 32 bits:

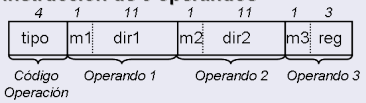
- 12 instrucciones de 3 operandos
  - Dos operandos, con direccionamiento absoluto e indirecto de memoria
  - Un operando, con direccionamiento de registro en indirecto de registro
- 150 instrucciones de 2 operandos
  - Un operando, con direccionamiento absoluto e indirecto de memoria
  - Un operando, con direccionamiento de registro en indirecto de registro
- 30 instrucciones de 0 operandos

Nota: *El campo dirección de los operandos con direccionamiento absoluto e indirecto de memoria será de 11 bits.*

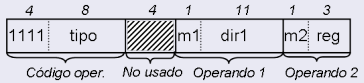
Ejemplo: Diseño de instrucción máquina

Solución

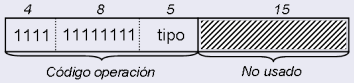
Instrucción de 3 operandos



Instrucción de 2 operandos



Instrucción sin operandos



Nomenclatura

mk: modo de direccionamiento del operando k
dirk: dirección del operando k
reg: número de registro
tipo: tipo de instrucción a ejecutar

Introducción

CISC

Complex Instruction Set Computer
Computador de conjunto de instrucciones complejo

- Desde los años 60 (en que aparece la unidad de control microprogramada) hasta principio de los años 80 la tendencia ha sido incrementar la complejidad de la CPU
  - Gran número de instrucciones en el repertorio e instrucciones complejas
  - Uso de gran número de modos de direccionamiento complejos

Introducción

Esta tendencia se debe a varias razones

- Aparecen lenguajes de programación de alto nivel cada vez más sofisticados
  - Diseño de instrucciones complejas que permitan implementar instrucciones de alto nivel directamente o con un pequeño número de instrucciones máquina
  - Una instrucción de alto nivel compleja implementable directamente en hardware, mejora la eficiencia del programa ya que se ejecuta mucho más rápidamente
- Gracias al uso de unidad de control microprogramada, se pueden implementar estas instrucciones máquinas complejas de forma simple, sin demasiado coste para el diseñador
- Facilidad para mantener la compatibilidad hacia abajo con los miembros de la misma familia
  - Añadir nuevas instrucciones al repertorio, manteniendo las antiguas

Problemas de las arquitecturas CISC

- Los estudios de ejecución de programas en arquitecturas CISC revelan que
  - La mayor parte de las instrucciones complejas apenas se utilizan en programas reales
  - Lo mismo ocurre con modos de direccionamiento complejos
- Dificultad para diseñar compiladores eficientes
  - Al aumentar la complejidad del repertorio se hace cada vez más difícil diseñar compiladores que aprovechen realmente esta gran variedad y versatilidad de instrucciones máquina

Problemas de las arquitecturas CISC

- Las instrucciones ocupan mucho espacio en memoria
  - Al existir un gran número de instrucciones y de modos de direccionamiento se necesita mucho espacio para codificar las instrucciones máquina y se tarda más tiempo en leerlas de memoria
  - El tiempo de acceso a memoria para captar instrucciones (Fetch) resulta cada vez más crítico conforme evolucionan los computadores

Debido a estos problemas a principio de los 80 surge la arquitectura RISC (Computador de conjunto de instrucciones reducido)
Idea básica: hacer rápidos y eficientes los casos más comunes a costa de reducir la velocidad en los casos menos comunes

Características

- Repertorio de instrucciones simple y ortogonal
  - Sólo se dispone de instrucciones máquina básicas
  - Repertorio ortogonal: no existen varias instrucciones distintas para realizar la misma operación
- Modos de direccionamiento sencillos
  - Cada operando sólo soporta unos pocos modos de direccionamiento simples
- Formatos de instrucción uniformes
  - Número reducido de formatos de instrucciones distintos
  - Todas las instrucciones de longitud similar

Características

- Tipos de datos simples
  - Las instrucciones sólo soportan unos pocos tipos de datos básicos distintos
- Arquitectura de tipo registro-registro
  - Sólo algunas instrucciones `MIPS: LOAD (Mem→Reg)` y `MIPS: STORE (Reg→Mem)` hacen referencia a memoria
  - El resto de instrucciones (aritméticas, lógicas, etc.) son de tipo registro-registro
    - Tanto los operandos fuente como destino son registros de la arquitectura
  - Las arquitecturas RISC suelen caracterizarse por disponer de un gran número de registros de propósito general

Ventajas de las arquitecturas RISC

- Su estructura es mucho más simple
  - Es más rápido y puede trabajar a mayor velocidad
  - Su diseño es más sencillo y barato
- Instrucciones más cortas
  - Tardan menos tiempo en leerse de memoria
- Mayor simplicidad para generar código máquina
  - Los compiladores son más eficientes y más sencillos de diseñar

Desventajas de las arquitecturas RISC

- El compilador genera un mayor número de instrucciones máquina
  - Se necesitan varias instrucciones para ejecutar las instrucciones de alto nivel
  - Mayor número de instrucciones $\Leftrightarrow$ Instrucciones más rápidas
- Más difícil mantener la compatibilidad entre computadores de la misma familia
  - Para diseñar un repertorio simple no se pueden 'heredar' todas las instrucciones

Computadores CISC

- VAX 11 (desaparecido)
- IBM Mainframes
  - serie 360, 370 y descendientes
- Intel 80x86
  - i8086, i80286, i80386, i80486, Pentium, Pentium II, ..., Pentium IV, IA64
- Motorola MC68xxx
  - MC68000, MC68010, MC68020, MC68030, MC68040, MC68060

Computadores RISC

- Motorola MC88000 (desaparecido)
- MIPS Rxxxx
  - R2000, R3000, R4000, R5000, R8000, R10000, R12000
- Sun SPARC
  - SPARC, Super SPARC 2, Ultra SPARC I, Ultra SPARC II, Ultra SPARC III
- HP PA-RISC
  - 7100, 7200, 7300, 8000, 8200, 8500
- PowerPC
  - 601, 602, 603, 604, 620, 630, Power3
- DEC Alpha
  - 21064, 21064a, 21066a, 21164, 21164a, 21264
- Intel 80860, Intel 80960

Ejemplo: MC68000
Características

- Procesador CISC de 16 bits
  - Aprox. 90 instrucciones máquina
  - 12 modos de direccionamiento
  - 9 formatos de instrucción distintos y con tamaños de una a cinco palabras
  - Ancho del bus de datos: 16 bits
  - Tamaño mínimo direccionable: 1 byte
  - Ancho del bus de direcciones: 24 bits (2^{24} bytes = 16 MB de memoria direccionables)

Ejemplo: MC68000
Modos de funcionamiento

- Modo usuario
  - Modo normal de ejecución de programas
  - No tiene acceso a determinadas instrucciones privilegiadas
  - No tiene acceso a determinados registros del supervisor
- Modo supervisor
  - Modo de ejecución del Sistema Operativo y durante el tratamiento de interrupciones
  - Tiene acceso a todas las instrucciones privilegiadas y a registros del supervisor

Ejemplo: MC68000
Tipos de datos y direccionamiento

- Tipos de datos: Visto en tema anterior
- Registros de la arquitectura: Visto en tema anterior
- Registro de estado: Visto en tema anterior
- Modos de direccionamiento: Visto en tema anterior

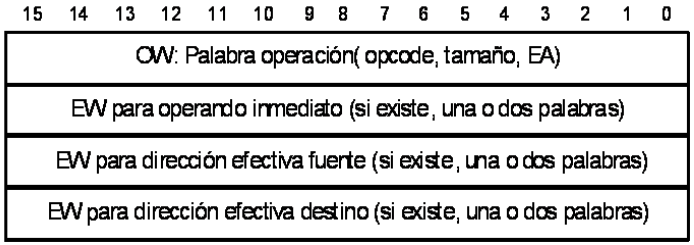
Ejemplo: MC68000
Formato de las instrucciones

- Características
  - Instrucciones de 0, 1 y 2 operandos
  - 9 formatos distintos
  - Longitud entre 1 y 5 palabras
- Contenidos de una instrucción máquina
  - Palabra operación (`OW`, siempre presente)
    - Primera palabra de la instrucción: Contiene el código de operación, modos de direccionamiento y tamaño de los operandos
  - Palabra de extensión (`EW`, de 0 a 4)
    - Información adicional de los operandos: Valor inmediato y desplazamiento
- Codificación modos de direccionamiento
  - Dos campos: modo y registro (6 bits)

3.1.2 Repertorio de instrucciones y formato de la instrucción máquina
Arquitecturas CISC y RISC
Ejemplos

Ejemplo: MC68000
Formato de las instrucciones

Formato genérico de la instrucción máquina



3.1.2 Repertorio de instrucciones y formato de la instrucción máquina
Arquitecturas CISC y RISC
Ejemplos

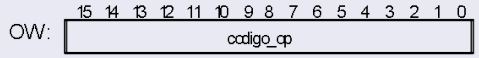
Ejemplo: MC68000
Formato de las instrucciones

	EA	
Codificación de los modos de direccionamiento	**modo**	**Registro**
Directo de reg. de datos	000	N° de reg. (Dn)
Directo de reg. de direcciones	001	N° de reg. (An)
Indirecto de reg.	010	N° de reg. (An)
Indirecto de reg. con postincr.	011	N° de reg. (An)
Indirecto de reg. con predecr.	100	N° de reg. (An)
Indirecto de reg. con desplaz.	101	N° de reg. (An)
Indirecto de reg. index. con deplaz.	110	N° de reg. (An)
Absoluto corto	111	000
Absoluto largo	111	001
Relativo al PC con deplaz.	111	010
Relativo al PC index. con deplaz.	111	011
Inmediato	111	100

3.1.2 Repertorio de instrucciones y formato de la instrucción máquina
Arquitecturas CISC y RISC
Ejemplos

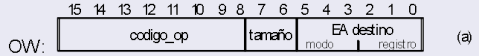
Ejemplo: MC68000
Instrucciones de 0 y 1 operandos

Instrucciones de 0 operandos



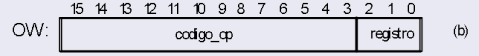
Ejemplos: *RTS*
NOP

Instrucciones de 1 operando



Campo tamaño	Tamaño operando
00	byte (.B)
01	palabra (.W)
10	doble palabra (.L)

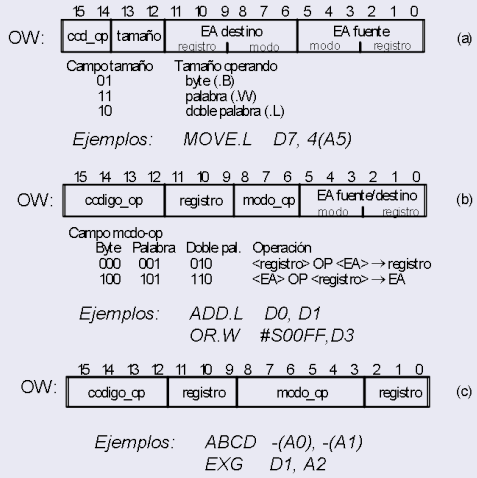
Ejemplos: *CLR.L D0*
NEG.B (A3)



Ejemplos: *SWAP R1*

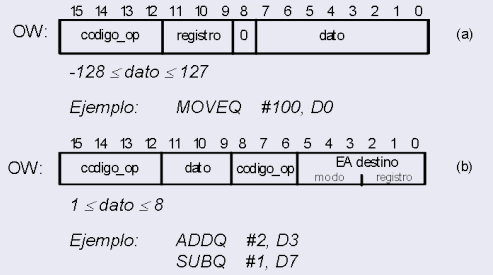
Ejemplo: MC68000
Instrucciones de 2 operandos

3 formatos

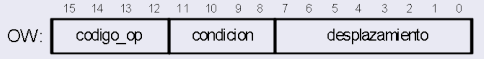


Ejemplo: MC68000
Otras

2 operandos con operando inmediato corto



Bifurcación condicional: Bcc



Ejemplo: BEQ dir1

Ejemplo: MC68000
Instrucciones de movimiento de datos

INSTR.	DESCRIPTION	EXAMPLE	

MOVE	Copies an 8-, 16- or 32-bit value from one memory location or register to another memory location or register	`MOVE.B #$8C,D0`
		`MOVE.W #$8C,D0`
		`MOVE.L #$8C,D0`
			`[D0]←$XXXXXX8C`
			`[D0]←$XXXX008C`
			`[D0]←$0000008C`

MOVEA	Copies a source operand to an address register. MOVEA operates only on words or longwords. MOVEA.W sign-extends the 16-bit operand to 32 bits.	`MOVEA.W #$8C00,A0` `MOVEA.L #$8C00,A0`	`[A0]←$FFFF8C00` `[A0]←$00008C00`
MOVEQ	Copies a 8-bit signed value in the range −128 to +127 to one of the eight data registers. The data to be moved is sign-extended before it is copied to its destination.	`MOVEQ #-3,D0` `MOVEQ #4,D0`	`[D0]←$FFFFFFFD` `[D0]←$00000004`
MOVEM	Transfers the contents of a group of registers specified by a list. The list of registers is defined as $A_i - A_j / D_p - D_q$. MOVEM operates only on words or longwords.	`MOVEM.L A0-A3/D0-D7,-(A7)`	`;copies all working` `;registers to stack`

Ejemplo: MC68000
Instrucciones de aritmética entera

INSTR.	DESCRIPTION	EXAMPLE	

ADDx
SUBx
	ADDx/SUBx add/subtract the contents of a source to/from the contents of a destination and deposits the result in the destination location. Direct memory-to-memory operations are not permitted. Assume `[D0]=$11118000` and `[D1]=$11110123`.	`ADD.W D0,D1`
		`ADD.L D0,D1`
		`ADDQ #N,D1`
		`SUB.L D1,D0`
			`[D1]←$11118123`
			`[D1]←$22228123`
			$N \in [1, 8]$
			`[D0]←$00007EDD`

MULU **MULS**	MULU (multiply unsigned) forms the product of two 16-bit integers. The 32-bit destination must be a data register. MULS is similar but treats data as signed. Assume `[D0]=$ABCD8000`.	`MULU #$0800,D0`	`[D0]←$00400000`
DIVU **DIVS**	DIVU (divide unsigned) works with a 32-bit dividend and a 16-bit divisor. The dividend must be a data register. The 16-bit result is stored in the low word of the destination, and the 16-bit remainder in the high word. DIVS is similar but treats data as signed. Assume `[D0]=$0000000E`, 14_{10}.	`DIVS #-3,D0`	`[D0]←$0002FFFC`

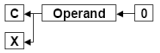
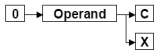
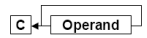
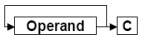
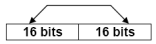
CLR
NEG
	CRL (clear) writes zeros into the destination operand. NEG (negate) performs a 2s complement operation on the destination data–subtracts it from zero. Assume `[D0]=$1234B021`.	`CLR.B D0`
		`CLR.L D0`
		`NEG.W D0`
			`[D0]←$1234B000`
			`[D0]←$00000000`
			`[D0]←$12344FDF`

EXT	Sign-extend increases the bit-size of a signed integer. EXT.W converts an 8-bit into a 16-bit, and EXT.L converts a 16-bit into a 32-bit. Assume `[D0]=$1234B021`.	`EXT.W D0` `EXT.L D0`	`[D0]←$12340021` `[D0]←$FFFFB021`

Ejemplo: MC68000
Instrucciones booleanas

INSTR.	DESCRIPTION	EXAMPLE	
AND **ANDI**	Bit-wise logical AND operation. Normally used to clear, or mask, certain bits in a destination operand.	`ANDI.B #%0111111,D0`	`clear the 8th least significant bit of D0`
OR **ORI**	Bit-wise logical OR operation. Normally used to set certain bits in a destination operand.	`ORI.B #%10101010,D0`	`set even bits of D0 lowest byte`
EOR **EORI**	Bit-wise logical XOR operation.	`EOR.B #%11111111,D0`	`XOR of the lowest byte of D0`
NOT	Bit-wise NOT operation. Assume `[D0]=$1234F0F0`.	`NOT.W D0`	`[D0]←$12340F0F`
TST	Similar to `CMP #0,operand`	`TST D0`	`update N,Z and clear V,C`

Ejemplo: MC68000
Instrucciones de desplazamiento y rotación

INSTR.	OPERATION	SCHEME
ASL	Arithmetic shift left	
ASR	Arithmetic shift right	
LSL	Logic shift left	
LSR	Logic shift right	
ROL	Rotate left	
ROR	Rotate right	
SWAP	Swap words of a long word	

Ejemplo: MC68000
Instrucciones de manipulación de bits

INSTR.	DESCRIPTION	EXAMPLE: Assume [D0]=$00000009	
BSET	Bit test and set Causes the Z-bit to be set if the specified bit is zero and then forces the specified bit of the operand to be set to one	`BSET #2, D0`	`[D0]←$0000000D, Z←1`
BCLR	Bit test and clear works like BSET except that the specified bit is cleared (forced to zero) after it has been tested	`BCLR #0, D0`	`[D0]←$00000008, Z←0`
BCHG	Bit test and change causes the value of the specified bit to be reflected in the Z-bit and then toggles (inverts) the state of the specified bit	`BCHG #4, D0`	`[D0]←$00000019, Z←1`
BTST	Bit test reflects the value of the specified bit in the Z-bit	`BTST #2, D0`	`Z←1`

Ejemplo: MC68000
Instrucciones BCD

INSTR.	DESCRIPTION	EXAMPLE: Assume [X]=0, [D0]=48, [D1]=21	
ABCD	Adds the source operand and the X-bit to the destination operand using BCD arithmetic. This is a BYTE operation only; the X-bit is used to provide a mechanism for multi-byte BCD operations.	`ABCD D0,D1`	`[D1]←$00000069`
SBCD	Subtract the source operand and the X-bit from the destination operand using BCD arithmetic. This is a BYTE operation only, so the X-bit is used to provide a mechanism for multi-byte BCD operations.	`SBCD D1,D0`	`[D0]←$00000027`
NBCD	Subtract the destination operand and the X-bit from zero.	`NBCD D1`	`[D1]←$00000052` `[X]←1, [V]←1, [C]←1`

Ejemplo: MC68000
Instrucciones de flujo

INSTR.	DESCRIPTION
BRA	BRA (branch always) implements an unconditional branch, relative to the PC. The offset is expressed as an 8- or 16-bit signed integer. If the destination is outside of a 16-bit signed integer, BRA cannot be used.
Bcc	Bcc (branch conditional) is used whenever program execution must follow one of two paths depending on a condition. The condition is specified by the mnemonic cc. The offset is expressed as an 8- or 16-bit signed integer. If the destination is outside of a 16-bit signed integer, Bcc cannot be used.
BSR,RTS	BSR branches to a subroutine. The PC is saved on the stack before loading the PC with the new value. RTS is use to return from the subroutine by restoring the PC from the stack.
JMP	JMP (jump) is similar to BRA. The only difference is that BRA uses only relative addressing, whereas JMP has more addressing modes, including absolute address (see reference manual).

cc	CONDITION	BRANCH TAKEN IF
CC	Carry clear	$C = 0$
CS	Carry set	$C = 1$
NE	Not equal	$Z = 0$
EQ	Equal	$Z = 1$
PL	Plus	$N = 0$
MI	Minus	$N = 1$
HI	Higher than	$\overline{CZ} = 1$
LS	Lower than or same as	$C + Z = 1$
GT	Greater than	$NV\overline{Z} + \overline{N}\overline{V}\overline{Z} = 1$
LT	Less than	$N\overline{V} + \overline{N}V = 1$
GE	Greater than or equal to	$NV + \overline{N}\overline{V} = 0$
LE	Less than or equal to	$Z + (NV + N\overline{V}) = 1$
VC	Overflow clear	$V = 0$
VS	Overflow set	$V = 1$
T	Always true	Always
F	Always false	Never

Ejemplo: MC68000
Instrucciones de control

INSTR.	DESCRIPTION
MOVE **ANDI** **ORI** **EORI**	Unique variations of MOVE, AND, OR and EOR that allow altering the bits in the status and condition code registers.
TRAP	TRAP performs three operations: (1) pushes the PC and SR to the stack, (2) sets the execution mode to supervisor and (3) loads the PC with a new value read from a vector table
STOP **RESET**	STOP loads the SR with an immediate operand and stops the CPU. RESET asserts the CPU's $\overline{RESET}$ line for 124 cycles. If STOP or RESET are executed in user mode, a privilege violation occurs.

Ejemplo: MIPS
Características

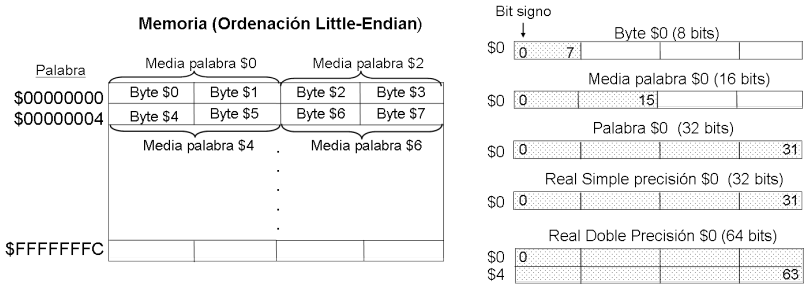
- Procesador RISC de 32 bits
- $\approx$ 70 instrucciones máquina
  - Repertorio ortogonal
  - Instrucciones clasificadas en cuatro grupos
    - Movimiento de datos
    - Aritmética entera, logicas y desplazamiento
    - Control de flujo
    - Aritmética en punto flotante
- 4 modos de direccionamiento
  - Inmediato
  - Directo de registros
  - Indirecto con desplazamiento
  - Indirecto con desplazamiento relativo al PC

Ejemplo: MIPS
Características

- Banco de 64 registros (32 bits cada uno)
  - 32 de propósito general (R0-R31)
  - 32 para instrucciones en punto flotante (F0-F31). Pueden usarse como:
    - 32 registros para operaciones en simple precisión (32 bits)
    - 16 registros para operaciones en doble precisión (64 bit)
- 3 formatos de instrucción distintos con longitud única de 32 bits
- Arquitectura registro-registro
  - Sólo las instrucciones de LOAD y STORE hacen referencia a memoria
  - El resto de instrucciones operan sobre registros
    - Instrucciones con tres operandos: 2 op. fuente y 1 op. Destino
    - Notación ensamblador: `op x,y,z` $\Leftrightarrow$ `x` $\leftarrow$ `(y) op (z)`

Ejemplo: MIPS
Tipos de datos

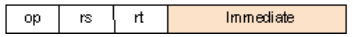
- Enteros
  - Tamaño Byte (8 bits)
  - Tamaño Media palabra (16 bits)
  - Tamaño Palabra (32 bits)
- Reales en punto flotante
  - Simple precisión (32 bits)
  - Doble precisión (64 bits)



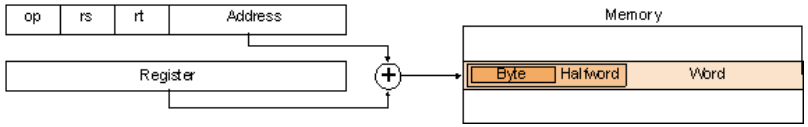
Ejemplo: MIPS
Modos de direccionamiento

- Directo de registro

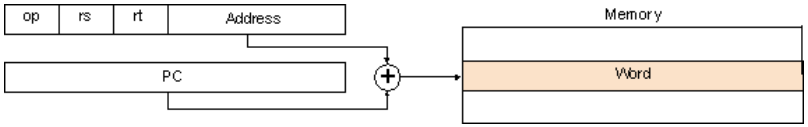
- Inmediato



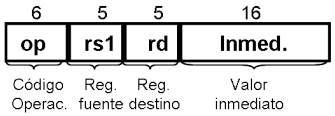
- Indirecto con desplazamiento



- Indirecto con desplazamiento relativo a PC



Ejemplo: MIPS
Formato de la instrucción máquina: Instrucción de tipo I



Instrucciones de LOAD/STORE

`LW $1,30($2)`	R1←Mem(30+(R2))
`SW 60($5),$10`	Mem(60+(R5))←(R10)

Instruc. con operando inmediato

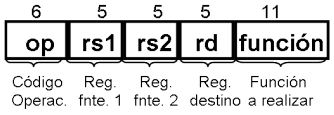
`ADDI $7,$9,#30`	R7←(R9)+30

Instruc. de bifurcación condicional

`BNE $5,$6,#0x0C`	if (R5≠R6) PC←0x0C

Ejemplo: MIPS
Formato de la instrucción máquina: Instrucción de tipo R

- Instruc. aritméticas, lógicas y desplazamiento de tipo de registro-registro
- El campo función especifica el tipo de operación a realizar: suma, resta, mult. div., and, or, desplaz., etc.



Ejemplos

`SUB $1,$2,$3`	R1←(R2)-(R3)
`SLL $1,$2,$3`	R1←(R2)<<(R3))

Ejemplo: MIPS

Formato de la instrucción máquina: Instrucción de tipo J

- Instrucciones de salto

Ejemplo

`J dir` PC←dir

Ejemplo: MIPS

Instrucciones de movimiento de datos

Instrucción	Significado	Ejemplo	Operación
LB	Load byte	`LB $2,40($3)`	$R2[24-31] \leftarrow Mem(40+(R3))_8$; $R1[0-23] \leftarrow (signo(Mem(40+(R3))))_{24}$
LBU	Load byte unsigned	`LBU $2,40($3)`	$R2[24-31] \leftarrow Mem(40+(R3))_8$; $R1[0-23] \leftarrow (0)_{24}$
LH	Load half word	`LH $2,40($3)`	$R2[16-31] \leftarrow Mem(40+(R3))_{16}$; $R1[0-15] \leftarrow (signo(Mem(40+(R3))))_{16}$
LHU	Load half word unsigned	`LHU $2,40($3)`	$R2[16-31] \leftarrow Mem(40+(R3))_{16}$; $R1[0-15] \leftarrow (0)_{16}$
LW	Load word	`LW $2,40($3)`	$R2[0-31] \leftarrow Mem(40+(R3))_{32}$
LF	Load Float	`LF $2,40($3)`	$F2[0-31] \leftarrow Mem(40+(R3))_{32}$
LD	Load double float	`LD $2,40($3)`	$F2[0-31], F3[0-31] \leftarrow Mem(40+(R3))_{64}$
SB	Store byte	`SB 40($3),$2`	$Mem(40+(R3))_8 \leftarrow R2[24-31]$
SH	Store half word	`SH 40($3),$2`	$Mem(40+(R3))_{16} \leftarrow R2[16-31]$
SW	Store word	`SW 40($3),$2`	$Mem(40+(R3))_{32} \leftarrow R2[0-31]$
SF	Store float	`SF 40($3),$2`	$Mem(40+(R3))_{32} \leftarrow F2[0-31]$
SD	Store double float	`SD 40($3),$2`	$Mem(40+(R3))_{64} \leftarrow F2[0-31], F3[0-31]$

Ejemplo: MIPS

Instrucciones de aritmética entera, lógicas y desplazamiento

Instrucción	Significado	Ejemplo	Operación
ADD/ADDU	Suma con/sin signo	ADD $1,$2,$3	$R1 \leftarrow (R2)+(R3)$
ADDI/ADDIU	Suma inmediato con/sin signo	ADDI $1,$2,#5	$R1 \leftarrow (R2)+5$
SUB/SUBU	Resta con/sin signo	SUB $1,$2,$3	$R1 \leftarrow (R2)-(R3)$
SUBI/SUBIU	Resta inmediato con/sin signo	SUBI $1,$2,#5	$R1 \leftarrow (R2)-5$
MULT/MULTU	Multiplicación con/sin signo	MULT $1,$2,$3	$R1 \leftarrow (R2) \times (R3)$
DIV/DIVU	División con/sin signo	DIV $1,$2,$3	$R1[0-15] \leftarrow Cociente((R2)/(R3))$ $R1[16-31] \leftarrow Resto((R2)/(R3))$
AND	Y lógica	AND $1,$2,$3	$R1 \leftarrow (R2) \wedge (R3)$
ANDI	Y lógica con inmediato	ANDI $1,$2,#5	$R1 \leftarrow (R2) \wedge 5$
OR	O lógica	OR $1,$2,$3	$R1 \leftarrow (R2) \vee (R3)$
ORI	O lógica con inmediato	ORI $1,$2,#5	$R1 \leftarrow (R2) \vee 5$
XOR	O exclusiva	XOR $1,$2,$3	$R1 \leftarrow (R2) \oplus (R3)$
XORI	O exclusiva con inmediato	XORI $1,$2,#5	$R1 \leftarrow (R2) \oplus 5$
SLL/SRL	Desplaz. lógico Izda./decha.	SLL $1,$2,$3	$R1 \leftarrow (R2) << (R3)$
SLLI/SRLI	Desp. Lóg. Izda./decha. con inmed.	SLLI $1,$2,#4	$R1 \leftarrow (R2) << 4$
SRA	Desplaz. aritmético derecha	SRA $1,$2,$3	$R1 \leftarrow (R2) << (R3)$ (mantiene signo)
SRAI	Desplaz. aritm. Dcha. con inmed.	SRA $1,$2,#4	$R1 \leftarrow (R2) << 4$ (mantiene signo)
Scc *	Set condicional	SLT $1,$2,$3	$if((R2) < (R3))R1 \leftarrow 1, else R1 \leftarrow 0$
Sccl *	Set condicional con inmediato	SEQI $1,$2,#0	$if((R2) == 0)R1 \leftarrow 1, else R1 \leftarrow 0$

(*) cc = LT, GT, LE, GE, EQ, NE

Ejemplo: MIPS

Otras

Instrucción de control de flujo

Instrucción	Significado	Ejemplo	Operación
J	Salto (jump)	`J dir`	$PC \leftarrow dir$
JR	Salto con registro	`JR $2`	$PC \leftarrow (R2)$
BEQZ	Bifurcación condic. si igual a cero	`BEQZ dir,$4`	$if((R4) == 0) PC \leftarrow (R4)$
BNEQZ	Bifurcación condic. si distinto de cero	`BNEQZ dir,$4`	$if((R4) \neq 0) PC \leftarrow (R4)$
JAL	Salto y link	`JAL dir`	$R31 \leftarrow (PC)+4; PC \leftarrow dir$
JALR	Salto y link con registro	`JALR dir,$2`	$R31 \leftarrow (PC)+4; PC \leftarrow (R2)$
TRAP	Provoca una excepción		
RFE	Retorno de excepción		

Instrucción de aritmética en punto flotante

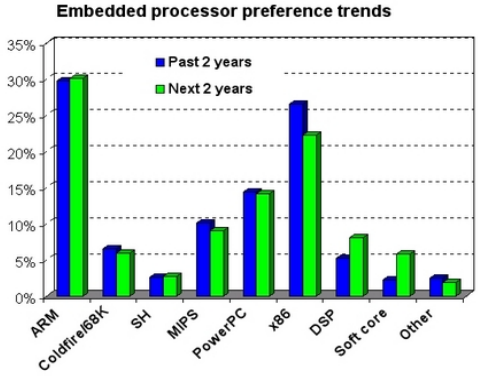
Instrucción	Significado
ADDF/ADDD	Suma en punto flotante simple/doble precisión
SUBF/SUBD	Resta en punto flotante simple/doble precisión
MULTF/MULTD	Multiplicación en punto flotante simple/doble precisión
DIVF/DIVD	División en punto flotante simple/doble precisión
CVTI2F/CVTI2D	Convierte entero a real simple precisión / doble precisión
CVTF2I/CVTF2D	Convierte real simple precisión a entero / real doble precisión
CVTD2I/CVTD2F	Convierte real doble precisión a entero / real simple precisión

Ejemplo: ARM
Origen

- ARM - Máquinas RISC Avanzadas (Advanced RISC Machines)
- Origen: Acorn RISC Machine, parte de un proyecto educativo (Inglaterra, 1984)
- 1987 - se convierte en el primer procesador RISC para uso comercial
- 1990 - escisión de Acorn Machines → ARM Ltd.
- ARM Ltd.: Creación de procesadores altamente configurables y adaptables a multitud de aplicaciones
- Innovaciones
  - Formato de instrucciones comprimido (thumb), con descompresión dinámica del flujo de instrucciones
  - Cauce segmentado (Pipeline) básico de 3 fases de gran simplicidad en el núcleo
  - Gran trabajo en temas de desarrollo y depurado software

Ejemplo: ARM
Mercado: Datos del 2007

- En el año 2005 ARM se estableció como líder del mercado (superando a Intel) en arquitecturas empotradas.
- El 30 % del mercado fue absorbido por ARM en el 2006 (frente al 28 % de Intel)



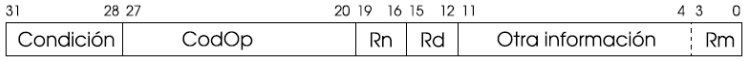
Ejemplo: ARM
Características

- ≈ 60 instrucciones máquina
  - Repertorio ortogonal
  - Instrucciones de 32 bits (salvo modo comprimido de 16 bits, thumb) alineadas en potencias de 4 en memoria
  - Clasificadas en 3 grupos
    - Instrucciones de proceso de datos: operaciones entre registros
    - Instrucciones de transferencia de datos: carga-almacenamiento
    - Instrucciones para el control del flujo
- Modos de direccionamiento (ver tema anterior)
  - Indexados con pre y post incremento
  - Desplazamiento relativos al PC

Ejemplo: ARM
Características

- Banco de 17 registros de 32 bits
  - 15 registros de propósito general (R0-R14)
    - R13: Puntero de pila
    - R14: Retorno de función
  - R15: Contador de programa
  - CPSR: Registro de estado
- Instrucciones fijas de 32 bits
- Arquitectura registro-registro
  - Sólo las instrucciones de LOAD y STORE hacen referencia a memoria
  - El resto de instrucciones operan sobre registros

Ejemplo: ARM
Formato de instrucción



- Condición: Código de ejecución condicional
  - De forma predeterminada el código indica ejecutar siempre
- CodOp: Código de operación
- Dos o tres registros (Rn, Rd y Rm)
  - Operaciones sobre registros:
    - Rd: Registro destino
    - Rn: Registro primer operando
    - Rm (opcional): Registro segundo operando
  - Operaciones registro-memoria (carga-almacenamiento):
    - Rd: Registro origen/destino
    - Rn: Registro base para construcción dirección de memoria
- Otra información adicional
  - Tamaño variable en función de la instrucción

Ejemplo: ARM
Ejemplos de instrucciones

Cód.	Nemón.	Interpretación	Estado necesario/Resultado
0000	EQ	Igual/igual a cero	$Z(1)$
0001	NE	Distinto	$Z(0)$
0110	VS	Desbordamiento	$V(1)$
0111	VC	No desbordamiento	$V(0)$
1000	HI	Mayor sin signo	$C(1)\&Z(0)$
1011	LT	Menor con signo	$N \neq V$
0000	AND	AND lógico bit-a-bit	$Rd \leftarrow Rn \wedge Op2$
0010	SUB	resta	$Rd \leftarrow Rn - Op2$
0011	RSB	resta al revés	$Rd \leftarrow Op2 - Rn$
0100	ADD	suma	$Rd \leftarrow Rn + Op2$
1010	CMP	comparar	cod. cond: $Rn - Op2$
1101	MOV	mover	$Rd \leftarrow Op2$
000	MUL	multiplicar	$Rd \leftarrow Rm \times Rs$

Ejemplo: ARM
Formato de instrucción en ensamblador

```
<codop>{<cond>}{S} Rd, Rn, operando2,{<desplazamiento>}
```

- codop: nemónico de la operación
- cond: sufijo opcional para ejecución condicional
- S: sufijo opcional para modificar códigos de condición (N,V,Z,C)
- Rd: Registro destino
- Rn: Registro del operando1
- operando2: Puede ser un tercer registro (Rm) o un operando inmediato
- desplazamiento: Desplazamiento (shift) opcional sobre operando2
  - tipo de desplazamiento + #cantidad de bits a desplazar