Arquitectura de Computadores II
Clase #3

Facultad de Ingeniería
Universidad de la República

Instituto de Computación
Curso 2010

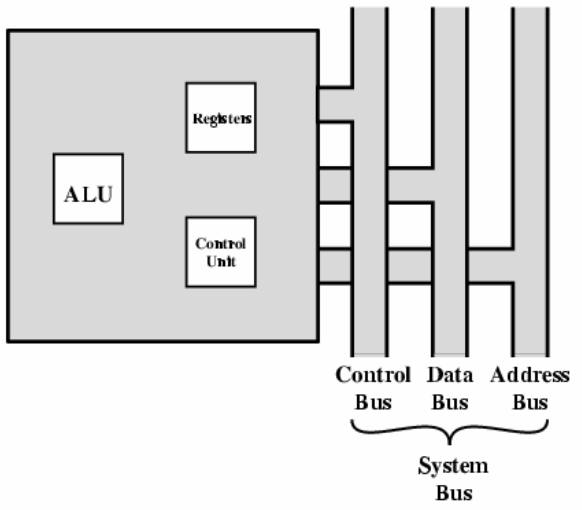
Veremos

- Registros
- Repertorio de instrucciones
- Modos de direccionamiento
  - El stack
- Formatos de datos
- Control de la CPU

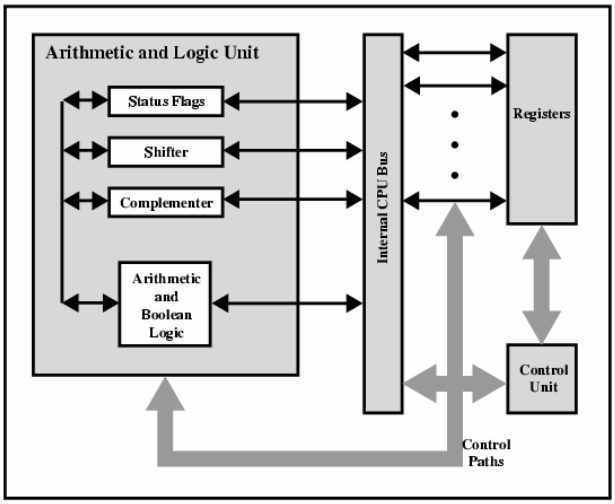
Estructura de la CPU

- Recordemos, la CPU debe:
  - Cargar instrucciones de memoria (Fetch)
  - Interpretar instrucciones
  - Procesar y transferir datos
- Se necesita almacenamiento temporal, los *registros*

La CPU y el bus del sistema



Estructura interna de la CPU



Registros

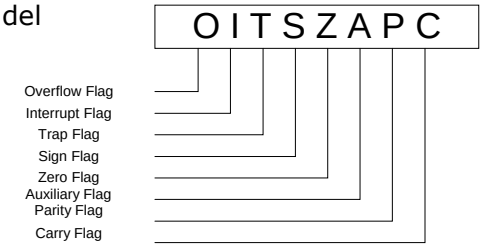
- La cantidad, tamaño y función de los registros varía según el diseño del procesador
- Es una de las decisiones más importantes del diseño
- Los registros constituyen el nivel superior de la jerarquía de memoria (próximas clases…)

Tipos de Registros

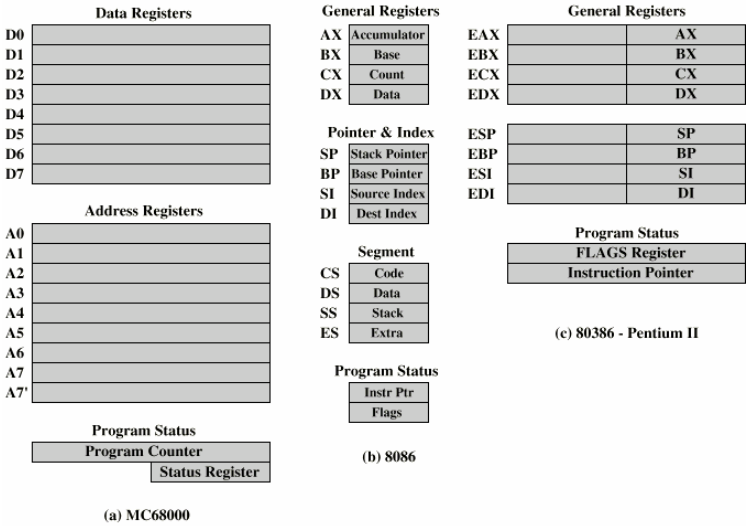
- Visibles al programador
  - Propósito general
    - Cuanto más generales, mayor flexibilidad para el usuario
    - Frecuentemente existen restricciones. Ej., la suma se aplica siempre sobre el registro AC.
  - Datos
    - operandos de ALU
  - Direcciones
    - segmento, puntero de stack, etc.
  - Códigos de condición
    - Se acceden implícitamente mediante instrucciones de salto condicional
    - En general forman parte de la *palabra de estado*
- Control y estado (PC, IR, MBR y MAR)
  - Algunos son visibles al programador y otros no
  - Ej. PC, IR, MBR, MAR, palabra de estado

Palabra de Estado (Program Status Word)

- Conjunto de bits de significado individual
- Códigos de Condición
  - Ej. el resultado de la última operación fue cero (Z)
- Lectura implícita por parte del programador
  - Ej. Jump if zero
- Usualmente NO se pueden setear explícitamente
- Otros bits
  - Interrupt enable/disable
  - Supervisor/usuario



Ejemplos de Organización de Registros

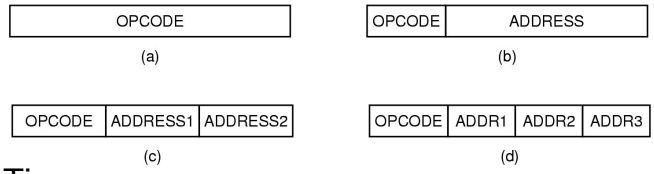


Repertorio de instrucciones (1/2)

- Es el conjunto de instrucciones distintas que puede ejecutar la CPU
- Elementos de una instrucción:
  - Código de operación
  - Referencia a operandos origen
  - Referencia a operandos destino
  - Referencia a la siguiente instrucción
- Operandos
  - Registros
  - Memoria
  - Entrada/Salida

Repertorio de instrucciones (2/2)

- Formato



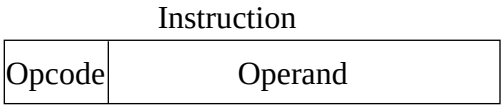
- Tipos
  - Procesamiento
  - Almacenamiento
  - Transferencia de datos
  - Control

Modos de direccionamiento (1/8)

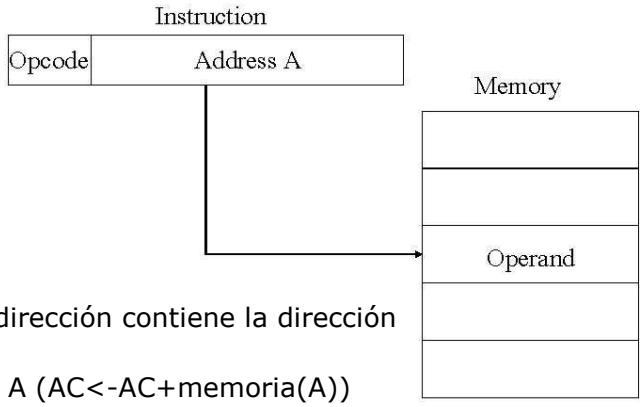
- Tipos
  - Inmediato
  - Directo
  - Indirecto
  - Registro
  - Indirecto con registro
  - Con desplazamiento
  - Pila

Modos de direccionamiento (2/8)

- Inmediato
  - El operando es parte de la instrucción
  - Ejemplo add 5 (AC <- AC + 5)

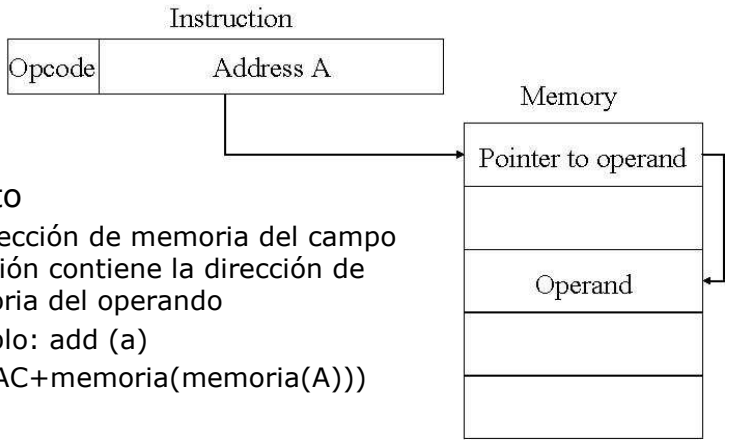


Modos de direccionamiento (3/8)



- Directo
  - El campo de dirección contiene la dirección de memoria
  - Ejemplo: add A (AC<-AC+memoria(A))

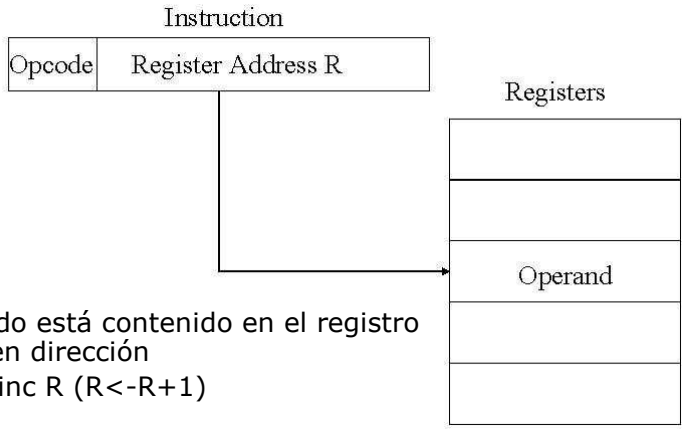
Modos de direccionamiento (4/8)



- Indirecto
  - La dirección de memoria del campo dirección contiene la dirección de memoria del operando
  - Ejemplo: add (a)

    (AC<-AC+memoria(memoria(A)))

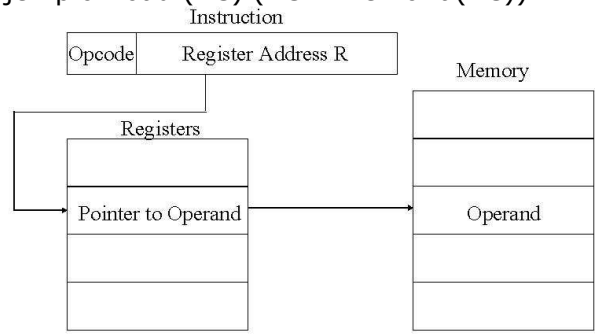
Modos de direccionamiento (5/8)



- Registro
  - El operando está contenido en el registro indicado en dirección
  - Ejemplo: inc R (R<-R+1)

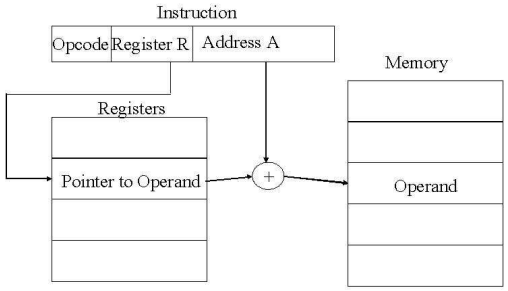
Modos de direccionamiento (6/8)

- Indirecto con registro
  - El operando está en la dirección contenida en el registro indicado en dirección
  - Ejemplo: load (AC) (AC<-memoria(AC))



Modos de direccionamiento (7/8)

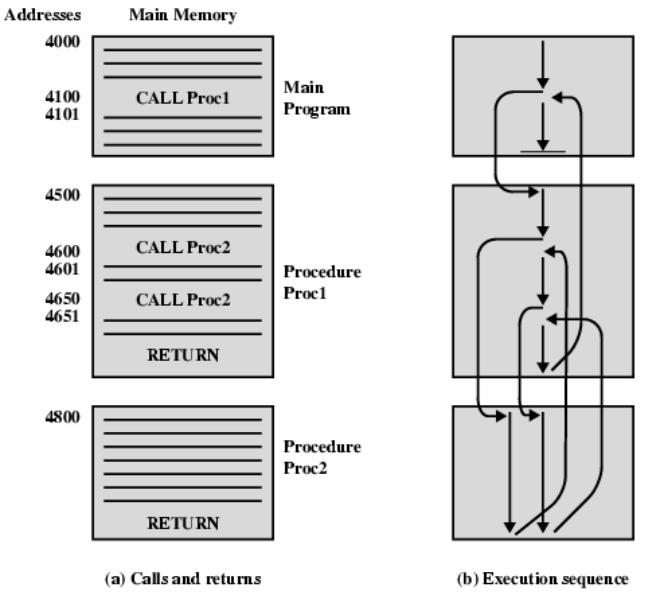
- Con Desplazamiento
  - Referencia a registro puede ser implícita o explícita
  - Usos
    - Relativo (jmp etiqueta, PC <- PC + etiqueta)
    - Con registro base (load (R+5), (AC<-memoria(R+5))
    - Indexado (mov R0, TABLA[R1])



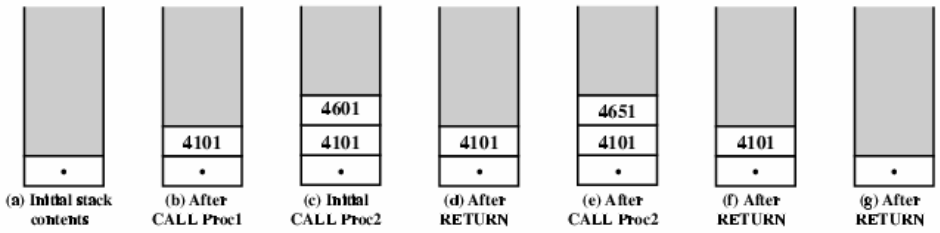
Modos de direccionamiento (8/8)

- Pila
  - En este caso se trabaja con operandos relativos al tope del stack
  - Ejemplo: add
    - pop(tmp1)
    - pop(tmp2)
    - tmp1<-- tmp1 + tmp2
    - push (tmp1)

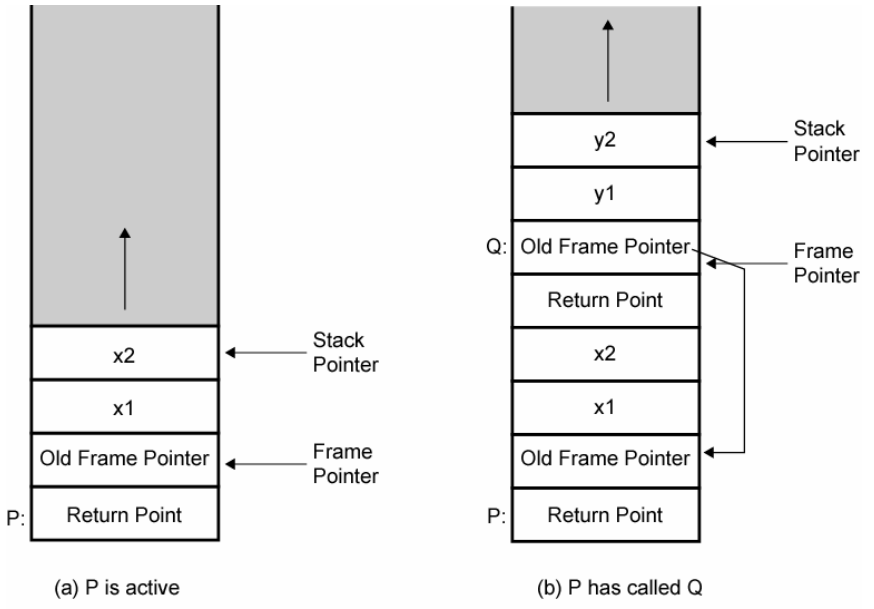
Llamadas anidadas



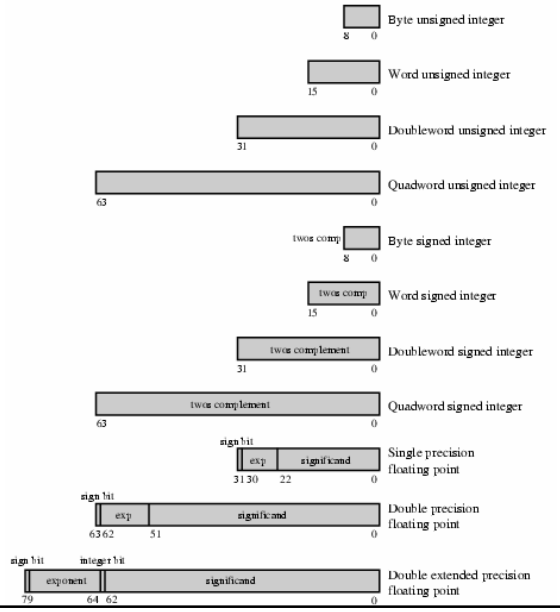
Uso del Stack (1/2)



Uso del Stack (2/2)



Tipos de Datos: ejemplo



Orden de los Bytes: Little-Endian y Big-Endian

- Indican de qué forma se almacena en memoria la secuencia de bytes que representan un **escalar** multi-byte
- *Little endian* indica que el byte menos significativo de la secuencia de bytes será almacenado en la dirección de memoria menor
- En la imagen se muestra cómo una secuencia de bytes, $byte_n$... $byte_0$, se guarda en memoria en cada caso. $byte_0$ es el menos significativo y $byte_n$ es el menos significativo

Dir	Big Endian	Little Endian
0	byte $_n$	byte $_0$
1	byte $_{n-1}$	byte $_1$
2	byte $_{n-2}$	byte $_2$
...		
n	byte $_0$	byte $_n$

Ejemplo (1/3)

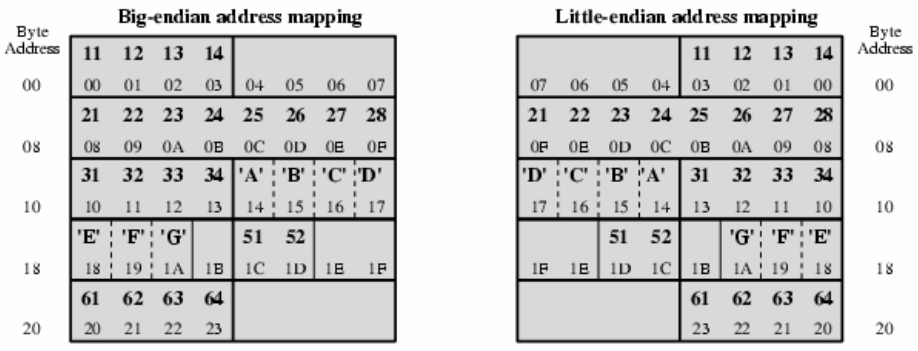
- Almacenar el hexa 12345678

Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
187	78	12

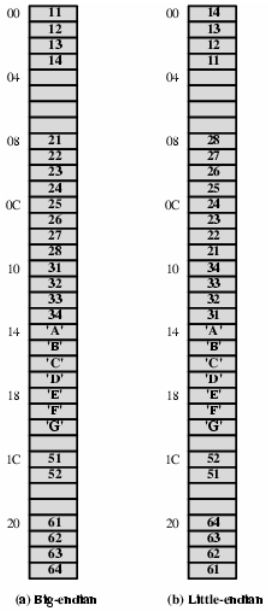
- Como se lee?

Ejemplo (2/3)

```
struct{
   int     a;     //0x1112_1314                  word
   int     pad;   //
   double  b;     //0x2122_2324_2526_2728        doubleword
   char*   c;     //0x3132_3334                  word
   char    d[7];  //'A','B','C','D','E','F','G'  byte array
   short   e;     //0x5152                       halfword
   int     f;     //0x6161_6364                  word
} s;
```



Ejemplo (3/3): otra visión del mapa de memoria



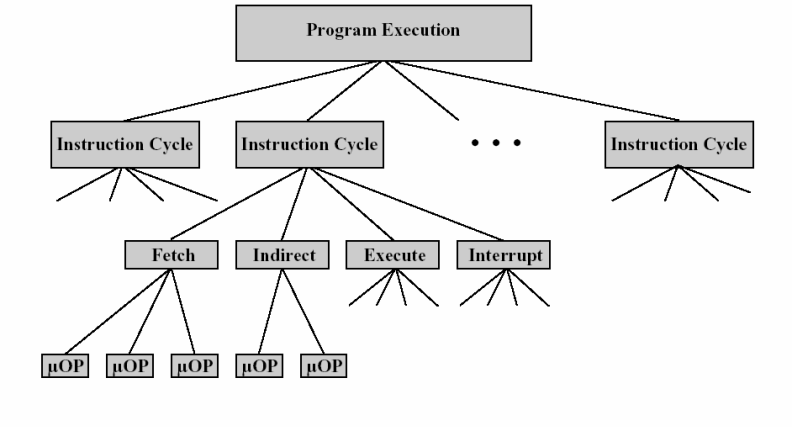
Endian…no hay estándar?!

- Pentium (80x86), VAX: little-endian
- IBM 370, Moterola 680x0 (Mac original), RISC: big-endian
- Internet: big-endian
  - Operaciones de sockets proveen funciones de conversión
    - htoi and itoh (Host to Internet & Internet to Host)

Control dentro de la CPU

- Sabemos que
  - Los programas se ejecutan como una secuencia de instrucciones
  - Cada instrucción se compone de una serie de pasos que constituyen el Ciclo de Instrucción
- Cada uno de estos pasos, a su vez, se desagrega en un serie de pequeños pasos denominados *microoperaciones*
- La unidad de control es responsable de generar esta secuencia de microoperaciones en el orden temporal adecuado
- La realización de estas microoperaciones genera flujo de datos entre diferentes componentes de la CPU

Ejecución de un programa

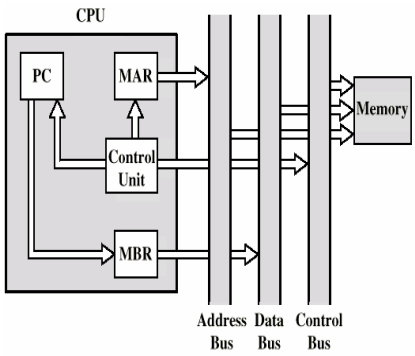


Registros

- Memory Address Register (MAR)
  - Conectado al bus de direcciones
  - Especifica direcciones para operaciones de lectura o escritura
- Memory Buffer Registry (MBR)
  - Conectado al bus de datos
  - Contiene datos a escribir o leídos
- Program Counter (PC)
  - Contiene la dirección de la próxima instrucción
- Instruccion Registry (IR)
  - Contiene la última instrucción cargada

Ciclo de Instrucción

- El ciclo de instrucción está compuesto por una serie de unidades más pequeñas:
  - Ciclo de fetch.
  - Ciclo indirecto.
  - Ciclo de ejecución.
  - Ciclo de interrupción.



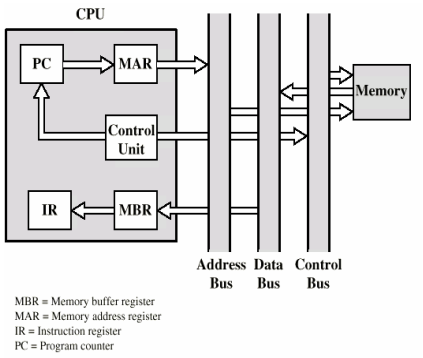
Ciclo de Fetch

- Obtiene una instrucción de memoria.
- Microoperaciones:

```
t1: MAR <-- PC
t2: MBR <-- memory
    PC  <-- PC+1
t3: IR  <-- MBR
```

o también

```
t1: MAR <-- PC
t2: MBR <-- memory
t3: PC  <-- PC+1
    IR  <-- MBR
```



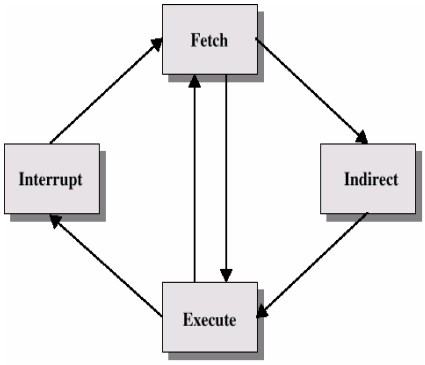
Ciclo Indirecto (i)

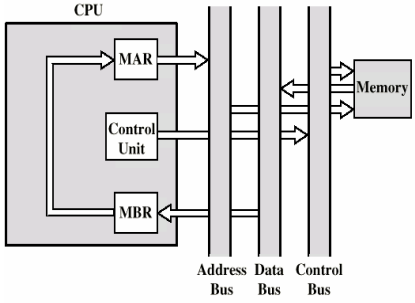
- Direccionamiento directo: un acceso a memoria por operando
- Direccionamiento Indirecto requiere más accesos a memoria
- Se puede tomar como un subciclo de instrucción adicional

- Microoperaciones:

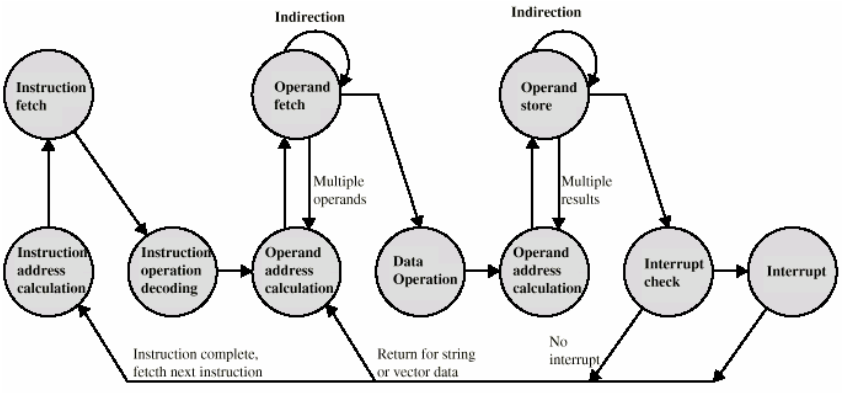
```
t1: MAR <-- IR(direccion)
t2: MBR <-- memory
t3: MAR <-- MBR
t4: MBR <-- memory
```

Ciclo Indirecto (ii)





Ciclo de Instrucción con indirección



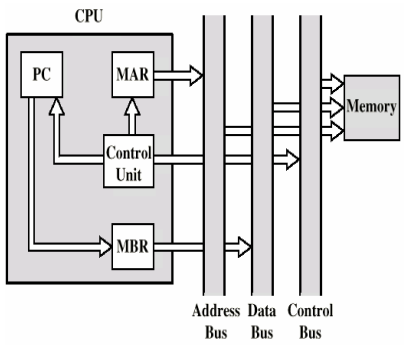
Ciclo de Interrupción (i)

- Luego del ciclo de ejecución se verifica la ocurrencia de una interrupción y si las interrupciones están habilitadas se ejecuta el ciclo de interrupción.
- Este ciclo varía mucho entre las distintas arquitecturas.
- Microoperaciones:

```
t1: MBR <-- PC
t2: MAR <-- dirección de respaldo PC
    PC  <-- dirección del manejador
t3: WRITE
```


Ciclo de Interrupción (ii)

- PC actual se salva para permitir retomar la ejecución después de la interrupción
- Contenido del PC se copia al MBR.
- Una posición de memoria especial (Ej. stack pointer) se carga al MAR
- MBR se escribe en memoria
- PC se carga con la dirección de la rutina de atención a la interrupción
- Fetch de la próxima instrucción (primera de la rutina de atención de la interrupción).



Ciclo de Ejecución (i)

- Puede ser muy variable
- Depende de la instrucción a ejecutar
- Puede incluir
  - Lectura/escritura de memoria
  - Entrada/Salida
  - Transferencia entre registros
  - Operaciones de la ALU

Ciclo de Ejecución (ii)

- Ejemplo 1
  - Instrucción: add R1, X
  - Microinstrucciones

```
t1: MAR <-- IR(direccion)
t2: READ
t3: R1  <-- R1 + MBR
```

- Ejemplo 2
  - Instrucción: ISZ X (Incrementar y saltar si es cero)
  - Microinstrucciones

```
t1: MAR <-- IR(direccion)
t2: READ
t3: MBR <-- MBR + 1
t4: WRITE
    Si MBR = 0 entonces PC <-- PC + 1
```

Preguntas