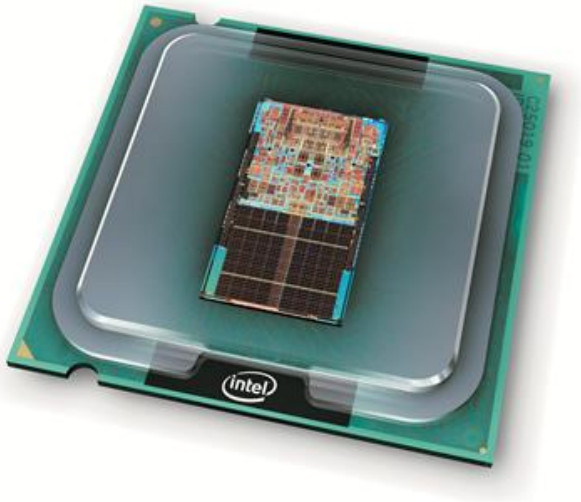


Bare Metal Development

Alejandro Furfaro

10 de mayo de 2019

Temario

1. Inicialización de un computador IA-32 desde el Reset
   - Arranque de un Procesador BSP (o único)
2. Boot de un Sistema Operativo
   - Análisis Conceptual de la tarea
3. Modo Protegido
   - Responde a un requerimiento
4. A20: La pesada herencia. . .
   - Compatibilidad con 8086

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro `eax` en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **`eax`** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **`eax`** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **`eax`** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **eax** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **eax** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **eax** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

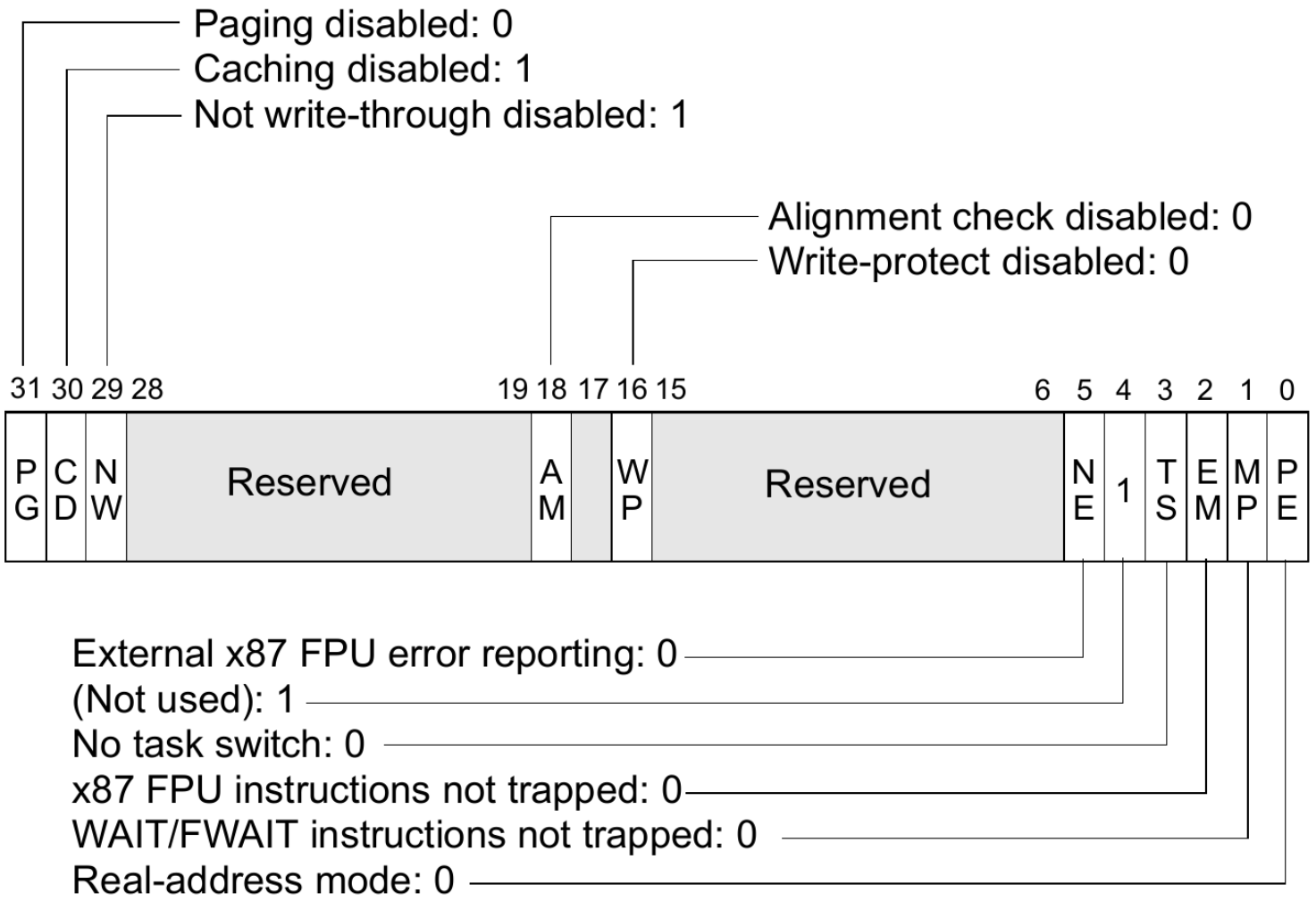
1. Built In Self Test

- Es un test interno que ejecuta el procesador luego de su encendido o cuando se activa su pin #RESET
- Como resultado inicializa el registro **eax** en 0x0 solo si todos los test pasaron OK.
- El resto de los registros toman un valor bien determinado
- Se invalidan las memorias cache internas
- Se invalidan las TLB[1]
- Se limpian los Branch Target Buffers[2]
- La diferencia con enviar una señal #INIT es que en éste caso, se mantienen intactos los valores de los Registros de la FPU, los MSR's y los MTRR's, y los caches internos.
- De acuerdo a la familia de procesador implementa con variantes la inicialización de los demás procesadores presentes en caso de un sistema SMP

[1] Ver Paginación de Memoria

[2] Ver Microarquitectura - Predicción de saltos

2. Valores iniciales de interés



Registro CR0. Valor inicial

2. Valores iniciales de interés

Register	Power up	Reset	INIT
EFLAGS[1]	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H[2]	60000010H[2]	60000010H[2]
CR2, CR3, CR4	00000000H	00000000H	00000000H
CS	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed
SS, DS, ES, FS, GS	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed
EDX	000n06xxH[3]	000n06xxH[3]	000n06xxH[3]
EAX	0[4]	0[4]	0[4]
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H

Valor iniciales de los registros corrientes[3]

[3] Para ampliar: Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a. Capítulo 9

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector

- Del slide anterior nos quedamos con los registros CS:IP, cuyos valores iniciales son respectivamente 0xF000:0xFFF0.
- Al mismo tiempo el procesador está en modo real de modo que su rango de direccionamiento físico es desde 0x00000 hasta 0xFFFFF, es decir 1Mbyte.
- de modo que la dirección física que generaría un procesaador 8086 viene dada pro la expresión: $cs << 4 + ip$
- Esto llevará al procesador a realizar el primer OPCODE FETCH en la dirección física 0xFFFF0.
- En este rango de direcciones es necesario mapear una memoria No volátil.

3. Reset Vector (Problemas asociados a esta herencia)

- Uno de nuestros principales problemas consiste en que el 8086 hace tiempo que no existe y en cambio sus descendientes compatibles poseen un rango de direccionamiento muchísimo mas amplio.
- Sin embargo por compatibilidad los procesadores posteriores (aún los modernos Corei7 con varios cores) arrancan en Modo Real accediendo en principio al espacio de direccionamiento de 1Mbyte que referimos anteriormente.
- Por lo tanto, necesitaríamos colocar una memoria no volátil alrededor del primer Mbyte, interrumpiendo la continuidad de los grandes bancos de DRAM que desde hace tiempo se comercializan.
- Intel pensó por lo tanto en una alternativa

3. Reset Vector (Problemas asociados a esta herencia)

- Uno de nuestros principales problemas consiste en que el 8086 hace tiempo que no existe y en cambio sus descendientes compatibles poseen un rango de direccionamiento muchísimo mas amplio.
- Sin embargo por compatibilidad los procesadores posteriores (aún los modernos Corei7 con varios cores) arrancan en Modo Real accediendo en principio al espacio de direccionamiento de 1Mbyte que referimos anteriormente.
- Por lo tanto, necesitaríamos colocar una memoria no volátil alrededor del primer Mbyte, interrumpiendo la continuidad de los grandes bancos de DRAM que desde hace tiempo se comercializan.
- Intel pensó por lo tanto en una alternativa

3. Reset Vector (Problemas asociados a esta herencia)

- Uno de nuestros principales problemas consiste en que el 8086 hace tiempo que no existe y en cambio sus descendientes compatibles poseen un rango de direccionamiento muchísimo mas amplio.
- Sin embargo por compatibilidad los procesadores posteriores (aún los modernos Corei7 con varios cores) arrancan en Modo Real accediendo en principio al espacio de direccionamiento de 1Mbyte que referimos anteriormente.
- Por lo tanto, necesitaríamos colocar una memoria no volátil alrededor del primer Mbyte, interrumpiendo la continuidad de los grandes bancos de DRAM que desde hace tiempo se comercializan.
- Intel pensó por lo tanto en una alternativa

3. Reset Vector (Problemas asociados a esta herencia)

- Uno de nuestros principales problemas consiste en que el 8086 hace tiempo que no existe y en cambio sus descendientes compatibles poseen un rango de direccionamiento muchísimo mas amplio.
- Sin embargo por compatibilidad los procesadores posteriores (aún los modernos Corei7 con varios cores) arrancan en Modo Real accediendo en principio al espacio de direccionamiento de 1Mbyte que referimos anteriormente.
- Por lo tanto, necesitaríamos colocar una memoria no volátil alrededor del primer Mbyte, interrumpiendo la continuidad de los grandes bancos de DRAM que desde hace tiempo se comercializan.
- Intel pensó por lo tanto en una alternativa

3. Reset Vector (Problemas asociados a esta herencia)

- Uno de nuestros principales problemas consiste en que el 8086 hace tiempo que no existe y en cambio sus descendientes compatibles poseen un rango de direccionamiento muchísimo mas amplio.
- Sin embargo por compatibilidad los procesadores posteriores (aún los modernos Corei7 con varios cores) arrancan en Modo Real accediendo en principio al espacio de direccionamiento de 1Mbyte que referimos anteriormente.
- Por lo tanto, necesitaríamos colocar una memoria no volátil alrededor del primer Mbyte, interrumpiendo la continuidad de los grandes bancos de DRAM que desde hace tiempo se comercializan.
- Intel pensó por lo tanto en una alternativa

3. Reset Vector (Recursos de modo protegido en modo real)

- Los registros de segmento en Modo Protegido trabajan con un registro cache para guardar sus descriptores a fin de evitar acceder a las tablas de descriptores cada vez que se necesita efectuar un acceso a memoria (o sea... siempre!).
- Todo lo que hacen los procesadores posteriores al 8086 es usarlos en modo real
- Y por lo tanto les asigna valores que resulten convenientes, para diseñar el mapa de memoria de un sistema x86 de manera mas sensata.
- Por lo tanto en el reset el estado de estos registros “amanece”como vemos a continuación:

3. Reset Vector (Recursos de modo protegido en modo real)

- Los registros de segmento en Modo Protegido trabajan con un registro cache para guardar sus descriptores a fin de evitar acceder a las tablas de descriptores cada vez que se necesita efectuar un acceso a memoria (o sea... siempre!).
- Todo lo que hacen los procesadores posteriores al 8086 es usarlos en modo real
- Y por lo tanto les asigna valores que resulten convenientes, para diseñar el mapa de memoria de un sistema x86 de manera mas sensata.
- Por lo tanto en el reset el estado de estos registros “amanece”como vemos a continuación:

3. Reset Vector (Recursos de modo protegido en modo real)

- Los registros de segmento en Modo Protegido trabajan con un registro cache para guardar sus descriptores a fin de evitar acceder a las tablas de descriptores cada vez que se necesita efectuar un acceso a memoria (o sea... siempre!).
- Todo lo que hacen los procesadores posteriores al 8086 es usarlos en modo real
- Y por lo tanto les asigna valores que resulten convenientes, para diseñar el mapa de memoria de un sistema x86 de manera mas sensata.
- Por lo tanto en el reset el estado de estos registros “amanece”como vemos a continuación:

3. Reset Vector (Recursos de modo protegido en modo real)

- Los registros de segmento en Modo Protegido trabajan con un registro cache para guardar sus descriptores a fin de evitar acceder a las tablas de descriptores cada vez que se necesita efectuar un acceso a memoria (o sea... siempre!).
- Todo lo que hacen los procesadores posteriores al 8086 es usarlos en modo real
- Y por lo tanto les asigna valores que resulten convenientes, para diseñar el mapa de memoria de un sistema x86 de manera mas sensata.
- Por lo tanto en el reset el estado de estos registros “amanece”como vemos a continuación:

3. Reset Vector (Recursos de modo protegido en modo real)

- Los registros de segmento en Modo Protegido trabajan con un registro cache para guardar sus descriptores a fin de evitar acceder a las tablas de descriptores cada vez que se necesita efectuar un acceso a memoria (o sea... siempre!).
- Todo lo que hacen los procesadores posteriores al 8086 es usarlos en modo real
- Y por lo tanto les asigna valores que resulten convenientes, para diseñar el mapa de memoria de un sistema x86 de manera mas sensata.
- Por lo tanto en el reset el estado de estos registros “amanece”como vemos a continuación:

3. Reset Vector - Recursos de MP en MR

		Base Address	Limit	Attrib
CS	0xF000	0xFFFF0000	0x0000FFFF	XX
DS	0x0000	0x00000000	0x0000FFFF	XX
ES	0x0000	0x00000000	0x0000FFFF	XX
SS	0x0000	0x00000000	0x0000FFFF	XX
FS	0x0000	0x00000000	0x0000FFFF	XX
GS	0x0000	0x00000000	0x0000FFFF	XX

Sumando a esto que el registro IP en el reset vale 0xFFF0, la dirección en la que el procesador aún en modo real irá a buscar su primer instrucción se resuelve mediante la siguiente expresión:

```
CS.BaseAddress + IP. = 0xFFFFFFF0
```

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

3. Reset Vector - Mapa de memoria inicial

- El procesador inicia su operación buscando operaciones en el fondo de la memoria. Puntualmente a 16 bytes de los 4Gbytes.
- Esto coincide con el anterior comportamiento de iniciar a 16 bytes del Mbyte del 8086.
- Es decir que el procesador inicia en Modo real con su restricción de espacio de direccionamiento a 1 Mbyte de memoria, pero puede realizar opcode fetch muy por encima de esa dirección. . . raro. . .
- Hay una condición para esto:

La letra chica del contrato :)

El valor del registro CS ***no puede cambiar mientras el procesador trabaje en modo real***. Ni bien lo haga, la Base Address de su registro cache se pone en 0x00000000, y automáticamente queda restringido a continuar buscando sus instrucciones en el primer Mega Byte de memoria.

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección de Modo

- En este punto es conveniente ocuparse de establecer el modo en que el procesador debe funcionar.
- Hay tres posibilidades.
  1. Dejarlo en modo Real
  2. Establecer Modo Protegido Flat
  3. Establecer Modo Protegido Segmentado (no recomendado para escribir firmware)
- Los modos 2 y 3 son variantes del Modo Protegido y tiene que ver con la forma en que organizaremos el modelo de segmentación

4. Selección del modo

- En la práctica la ROM se mapea físicamente en el fondo de los 4 Gbytes.
- Por ejemplo: Si usamos una EEPROM de 4 Kbytes (lo mas pequeño que popdemos conseguir actualmente en el mercado), el rango de direcciones para las que se activará el la línea Chip Select de este componente será: 0xFFFFF000 - 0xFFFFFFFF.
- Sin embargo cuando empecemos a escribir el programa le vamos a indicar al ensamblador que trabajamos en modo real y por lo tanto en el rango de direcciones 0xFF000-0xFFFFF.
- El ensamblador solo construirá un bloque de 4 Kbytes de código con una visión de un mapar de memoria de 1 Mbyte de capacidad y lu bicará en los 4 K finales. El decodificador de memoria de hardware lo ubicará en 0xFFFFF000.

4. Selección del modo

- En la práctica la ROM se mapea físicamente en el fondo de los 4 Gbytes.
- Por ejemplo: Si usamos una EEPROM de 4 Kbytes (lo mas pequeño que popdemos conseguir actualmente en el mercado), el rango de direcciones para las que se activará el la línea Chip Select de este componente será: 0xFFFFF000 - 0xFFFFFFFF.
- Sin embargo cuando empecemos a escribir el programa le vamos a indicar al ensamblador que trabajamos en modo real y por lo tanto en el rango de direcciones 0xFF000-0xFFFFF.
- El ensamblador solo construirá un bloque de 4 Kbytes de código con una visión de un mapar de memoria de 1 Mbyte de capacidad y lu bicará en los 4 K finales. El decodificador de memoria de hardware lo ubicará en 0xFFFFF000.

4. Selección del modo

- En la práctica la ROM se mapea físicamente en el fondo de los 4 Gbytes.
- Por ejemplo: Si usamos una EEPROM de 4 Kbytes (lo mas pequeño que popdemos conseguir actualmente en el mercado), el rango de direcciones para las que se activará el la línea Chip Select de este componente será: 0xFFFFF000 - 0xFFFFFFFF.
- Sin embargo cuando empecemos a escribir el programa le vamos a indicar al ensamblador que trabajamos en modo real y por lo tanto en el rango de direcciones 0xFF000-0xFFFFF.
- El ensamblador solo construirá un bloque de 4 Kbytes de código con una visión de un mapar de memoria de 1 Mbyte de capacidad y lu bicará en los 4 K finales. El decodificador de memoria de hardware lo ubicará en 0xFFFFF000.

4. Selección del modo

- En la práctica la ROM se mapea físicamente en el fondo de los 4 Gbytes.
- Por ejemplo: Si usamos una EEPROM de 4 Kbytes (lo mas pequeño que popdemos conseguir actualmente en el mercado), el rango de direcciones para las que se activará el la línea Chip Select de este componente será: 0xFFFFF000 - 0xFFFFFFFF.
- Sin embargo cuando empecemos a escribir el programa le vamos a indicar al ensamblador que trabajamos en modo real y por lo tanto en el rango de direcciones 0xFF000-0xFFFFF.
- El ensamblador solo construirá un bloque de 4 Kbytes de código con una visión de un mapar de memoria de 1 Mbyte de capacidad y lu bicará en los 4 K finales. El decodificador de memoria de hardware lo ubicará en 0xFFFFF000.

4. Selección del modo

- En la práctica la ROM se mapea físicamente en el fondo de los 4 Gbytes.
- Por ejemplo: Si usamos una EEPROM de 4 Kbytes (lo mas pequeño que popdemos conseguir actualmente en el mercado), el rango de direcciones para las que se activará el la línea Chip Select de este componente será: 0xFFFFF000 - 0xFFFFFFFF.
- Sin embargo cuando empecemos a escribir el programa le vamos a indicar al ensamblador que trabajamos en modo real y por lo tanto en el rango de direcciones 0xFF000-0xFFFFF.
- El ensamblador solo construirá un bloque de 4 Kbytes de código con una visión de un mapar de memoria de 1 Mbyte de capacidad y lu bicará en los 4 K finales. El decodificador de memoria de hardware lo ubicará en 0xFFFFF000.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensambador.
- No se genera código. Simplemente son directivas para el ensamblador.

- El único código para iniciar es un loop infinito.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensamba-dor.
- No se genera código. Simplemente son directivas para el ensam-blador.
- El único código para iniciar es un loop infinito.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensambador.
- No se genera código. Simplemente son directivas para el ensamblador.
- El único código para iniciar es un loop infinito.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensamba-dor.
- No se genera código. Simplemente son directivas para el ensam-blador.

```
ORG 0xFF000  ;Esto es: (1MB-4KB) -> 0x100000-0x1000=0xFF000.
USE16        ;Indica al ensamblador generar código de 16 bits
```

- El único código para iniciar es un loop infinito.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensambador.
- No se genera código. Simplemente son directivas para el ensamblador.

```
ORG 0xFF000  ;Esto es: (1MB-4KB) -> 0x100000-0x1000=0xFF000.
USE16        ;Indica al ensamblador generar código de 16 bits
```

- El único código para iniciar es un loop infinito.

5. Primer iagen de 4K

- En primer lugar hay que ponerle un marco de trabajo al ensambador.
- No se genera código. Simplemente son directivas para el ensamblador.

```
ORG 0xFF000 ;Esto es: (1MB-4KB) -> 0x100000-0x1000=0xFF000.
USE16       ;Indica al ensamblador generar código de 16 bits
```

- El único código para iniciar es un loop infinito.

```
init16:
      cli         ;Deshabilita interrupciones
      jmp init16  ;loop infinito
align 16          ;Completa hasta la siguiente dirección múltiplo
                  ;de 16 (verificar con que completa con NOP's).
```

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que **`init16`** efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que **`init16`** efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que `init16` efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que **`init16`** efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que **`init16`** efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

5.1. Completando la imagen de 4Kbytes

- Necesitamos que el label **`init16`** sea mapeado por el ensamblador exactamente en la dirección del reset vector: **`0xFFFF0`**.
- la sentencia align 16 vista solo nos asegura que el código generado a partir de **`init16`**, se complete con **`NOP`**'s hasta la dirección previa a la siguiente dirección alineada a 16 bits. Es decir, nos segura alinear el final de los 4Kbytes.
- Necesitamos completar los primeros 4080 bytes de los 4096 totales de modo que **`init16`** efectivamente ocupe la dirección `0xFFFF0`.
- Primer impulso:

```
    TIMES 4080 db 0x90   ;Generamos 4080 NOP's
init16:
      cli            ;Deshabilita interrupciones
      jmp init16     ;loop infinito
align 16             ;Completa hasta la siguiente dirección múltiplo
                     ;de 16 (verificar con que completa con NOP's).
```

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.1 Completando la imagen de 4Kbytes

- El código anterior resuelve el problema pero no es escalable.
- A medida que agreguemos código pra hacer algo mínimamente útil, deberemos ajustar ell parámetro de TIMES para reducir de 4080 a la cantidad que haga falta de acierdo con el tamaño del código que incluyamos.
- En realidad lo mas complejo será llevar la cuenta del tamaño del código
- Por lo tanto hay que buscar un método mas eficiente.
- En el siguiente código EQU (por Equal) le genera una constante al ensamblador cuyo valor calcula automáticamente el tamaño del código generado y lo resta del tamaño de la ROM. Problema resuelto.

5.2. Versión final de nuestro primer intento

```
ORG 0xFF000 ;Esto es: (1MB-4KB) -> 0x100000-0x1000=0xFF000.
USE16       ;Indica al ensamblador generar código de 16 bits

code_size EQU (end - init16) ; siempre es el tamaño del código

; Rellenamos la ROM con 0x90 (NOP)
times (4096-code_size) db 0x90

init16:
    cli         ;Deshabilita interrupciones
    jmp init16  ;loop infinito
aqui:
    hlt         ;Si por algún motivo sale del loop: HALT
    jmp aqui    ;Solo sale por reset o por interrupción
align 16        ;Completa hasta la siguiente dirección múltiplo
                ;de 16 (verificar con que completa con NOP's).
end:
```

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional).
  2. Inicializar el soporte del procesador para el manejo de memoria[5].
  3. Inicializar el chipset[6].

[4] Ampliar desde Section 9.11 | Microcode Update Facilities | In Intel 64 and IA-32 Architectures Software Developer's Manual Vol 3a.

[5] Ampliar desde Section 11.1 | Memory Types | In Intel 64 and IA-32 Architectures Software Developer's Manual Vol 3a.

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional)
  2. Inicializar el soporte del procesador para manejo de memoria[5]
  3. Inicializar el chipset[5]

[4] Ampliar desde Seccion 9.11 Microcode Update Facilities, Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a.

[5] BIOS/Firmware Writer's Guide (o Black Magic Manu

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional)
  2. Inicializar el soporte del procesador para manejo de memoria[5]
  3. Inicializar el chipset[5]

[4] Ampliar desde Seccion 9.11 Microcode Update Facilities, Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a.

[5] BIOS/Firmware Writer's Guide (a Black Magic Manual)

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional)
  2. Inicializar el soporte del procesador para manejo de memoria[5]
  3. Inicializar el chipset[5]

[4] Ampliar desde Seccion 9.11 Microcode Update Facilities, Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a.

[5] BIOS/Firmware Writer's Guide (a Black Magic Manual)

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional)
  2. Inicializar el soporte del procesador para manejo de memoria[5]
  3. Inicializar el chipset[5]

[4] Ampliar desde Seccion 9.11 Microcode Update Facilities, Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a.

[5] BIOS/Firmware Writer's Guide (a Black Magic Manual)

6. Preparar la Inicialización de memoria

- Todo el código que se ha ejecutado hasta este momento, se ha ejecutado desde NVRAM o sea Flash.
- Cuanto antes pasemos a ejecutar desde memoria DRAM, antes aceleraremos el inicio del sistema ya que la memoria DRAM es mas veloz que la Flash.
- Entonces es urgente inicializar la DRAM. Para ello hay que:
  1. Actualizar el microcódigo del procesador[4] (opcional)
  2. Inicializar el soporte del procesador para manejo de memoria[5]
  3. Inicializar el chipset[5]

[4] Ampliar desde Seccion 9.11 Microcode Update Facilities, Intel®64 and IA-32 Architectures Software Developer's Manual. Vol 3a.

[5] BIOS/Firmware Writer's Guide (a Black Magic Manual)

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory Initialization Reference Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory Initialization Reference Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory initialization Referencce Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory initialization Referencce Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory initialization Referencce Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory initialization Referencce Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

6. Inicialización de Memoria

- Otra caja negra.
- Intel provee (NDA mediante) los siguientes recursos:
  1. BIOS/Firmware Writer's Guide
  2. Memory initialization Referencce Code (MRC)
- Es chipset (plataforma) dependiente
- MRC puede estar escrito para ejecutar en código de 16 bits o 32 bits (Segmentación Flat o multisegmento).

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re Direction
  4. Configurar un Stack
  5. Transferir control de progrema a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7. Inicialización Post Memoria

- Una vez ejecutado el código que inicializa el controlador de memoria y los configura de acuerdo a los DIMMs detectados en el sistema,se termina el trabajo de inicialización de memoria.
- Es una secuencia de pasos adicional, pero hecha por nosotros
  1. Memory test
  2. Shadow Firmware
  3. Memory Transaction Re-Direction
  4. Configurar un Stack
  5. Transferir control de programa a DRAM

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.1. Memory Test

- Es un primer uso de la memoria para comprobar su integridad una vez zinicializado el controlador lo cual nos permite el acceso
- Puede estar incluido en el MRC (algunos vendors lo proveen)
- Es conveniente chequear memoria en este punto ya que los errores se producen de manera aleatoria e inconsistente.
- Simplifica el debug del firmware ya que si pasó el test, los errores posteriores no son atribuibles a memoria.
- Los test son un balance entre la granularidad (cada cuantos bytes se comprueba) vs. performance (en móviles el tiempo de arranque es importante)
- No hay algoritmos específicos, sino que son mas bien simples, y pueden hacerse en assembler.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Firmware Shadow

- Es código que se autocopia desde NVRAM (Lenta, típicamente es FLASH) a DRAM (mas rápida y por lo tanto mejora el tiempo del setup.
- Al habilitar el cache, cada acceso a la Flash (NVRAM) genera un Miss y retrasa enormemente la ejecución.
- Copiarlo a DRAM no solo acelera por ser esta mas rápida que la NVRAM, sino por ser cacheable.
- En los sistemas PC la zona de shadow debe copiarse desde la NVRAM hasta la zona por debajo del Mbyte.
- En otros sistemas el espacio de direccionamiento destino de la copia es arbitraria.

7.2. Memory Transaction Re-Direction

- Los chipsets generalmente están provistos de Registros PAM (Programmable Attribute Maps)
- Estos PAMs permiten controlar el aliasing de direcciones que hace posible leer o escribir secciones de memoria por debajo de 1 Mbyte hacia o desde DRAM o NVRAM cuyas direcciones rondan los 4 Gbytes.
- Antes de ejecutar shadowing puede requerirse acceder a estos registros.
- Los accesos dependen de cada chipset. Algunos permiten leer y escribir, otros solo leer.

7.2. Memory Transaction Re-Direction

- Los chipsets generalmente están provistos de Registros PAM (Programmable Attribute Maps)
- Estos PAMs permiten controlar el aliasing de direcciones que hace posible leer o escribir secciones de memoria por debajo de 1 Mbyte hacia o desde DRAM o NVRAM cuyas direcciones rondan los 4 Gbytes.
- Antes de ejecutar shadowing puede requerirse acceder a estos registros.
- Los accesos dependen de cada chipset. Algunos permiten leer y escribir, otros solo leer.

7.2. Memory Transaction Re-Direction

- Los chipsets generalmente están provistos de Registros PAM (Programmable Attribute Maps)
- Estos PAMs permiten controlar el aliasing de direcciones que hace posible leer o escribir secciones de memoria por debajo de 1 Mbyte hacia o desde DRAM o NVRAM cuyas direcciones rondan los 4 Gbytes.
- Antes de ejecutar shadowing puede requerirse acceder a estos registros.
- Los accesos dependen de cada chipset. Algunos permiten leer y escribir, otros solo leer.

7.2. Memory Transaction Re-Direction

- Los chipsets generalmente están provistos de Registros PAM (Programmable Attribute Maps)
- Estos PAMs permiten controlar el aliasing de direcciones que hace posible leer o escribir secciones de memoria por debajo de 1 Mbyte hacia o desde DRAM o NVRAM cuyas direcciones rondan los 4 Gbytes.
- Antes de ejecutar shadowing puede requerirse acceder a estos registros.
- Los accesos dependen de cada chipset. Algunos permiten leer y escribir, otros solo leer.

7.2. Memory Transaction Re-Direction

- Los chipsets generalmente están provistos de Registros PAM (Programmable Attribute Maps)
- Estos PAMs permiten controlar el aliasing de direcciones que hace posible leer o escribir secciones de memoria por debajo de 1 Mbyte hacia o desde DRAM o NVRAM cuyas direcciones rondan los 4 Gbytes.
- Antes de ejecutar shadowing puede requerirse acceder a estos registros.
- Los accesos dependen de cada chipset. Algunos permiten leer y escribir, otros solo leer.

7.3. Establecimiento de un stack

- Antes de saltar a memoria es necesario definir un stack
- Es necesario definir la cantidad de memoria y considerar que el stack crece hacia direcciones bajas.
- En modo 16 bits se configuran los registros `ss:sp`
- En modo Protegido (ya sea Flat o Segmentado) se configuran `ss:esp`

7.3. Establecimiento de un stack

- Antes de saltar a memoria es necesario definir un stack
- Es necesario definir la cantidad de memoria y considerar que el stack crece hacia direcciones bajas.
- En modo 16 bits se configuran los registros `ss:sp`
- En modo Protegido (ya sea Flat o Segmentado) se configuran `ss:esp`

7.3. Establecimiento de un stack

- Antes de saltar a memoria es necesario definir un stack
- Es necesario definir la cantidad de memoria y considerar que el stack crece hacia direcciones bajas.
- En modo 16 bits se configuran los registros `ss:sp`
- En modo Protegido (ya sea Flat o Segmentado) se configuran `ss:esp`

7.3. Establecimiento de un stack

- Antes de saltar a memoria es necesario definir un stack
- Es necesario definir la cantidad de memoria y considerar que el stack crece hacia direcciones bajas.
- En modo 16 bits se configuran los registros **`ss:sp`**
- En modo Protegido (ya sea Flat o Segmentado) se configuran **`ss:esp`**

7.3. Establecimiento de un stack

- Antes de saltar a memoria es necesario definir un stack
- Es necesario definir la cantidad de memoria y considerar que el stack crece hacia direcciones bajas.
- En modo 16 bits se configuran los registros **`ss:sp`**
- En modo Protegido (ya sea Flat o Segmentado) se configuran **`ss:esp`**

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

7.4. Salto a memoria DRAM

- Aquí es donde se puede producir un problema si no se hicieron bien las cosas.
- El programa debe haberse autocopiado de NVRAM a DRAM
- El salto se implementa mediante un salto FAR.
- En modo 16 bits o real hay que saltar a una dirección dentro del primer Mbyte de DRAM.
- En modo Protegido (ya sea Flat o Segmentado) no hay restricciones respecto de la dirección de destino, dentro de los 4 Gbytes de capacidad de direccionamiento. Pero debe estar el sistema correctamente inicializado (tablas de descriptores de segmento mínimas y demás delicias de la vida).

8. Habilitaciones de dispositivos misceláneos

- Este ítem es plataforma dependiente y se compone de un conjunto de dispositivos que surgen del análisis de lso esquemáticos de hardware.
- Los típicos dispositivos que están presentes en todos los chipsets (con las variantes de implementación de cada caso) son:
  1. Programación del chip de reloj
  2. GPIO: igualmente debe consultarse el Chipset EDS (Intel Under Guide (NDA mediante) para tener los detalles

8. Habilitaciones de dispositivos misceláneos

- Este ítem es plataforma dependiente y se compone de un conjunto de dispositivos que surgen del análisis de Iso esquemáticos de hardware.
- Los típicos dispositivos que están presentes en todos los chipsets (con las variantes de implementación de cada caso) son:
  1. Programación del chip de reloj
  2. GPIO: igualmente debe consultarse el Chipset EDS (Intel Corp.) (NDA mediante) para tener los detalles

8. Habilitaciones de dispositivos misceláneos

- Este ítem es plataforma dependiente y se compone de un conjunto de dispositivos que surgen del análisis de Iso esquemáticos de hardware.
- Los típicos dispositivos que están presentes en todos los chipsets (con las variantes de implementación de cada caso) son:
  1. Programación del chip de reloj
  2. GPIO. Igualmente debe consultarse el Chipset BIOS Writer Guide (NDA mediante) para tener los detalles.

8. Habilitaciones de dispositivos misceláneos

- Este ítem es plataforma dependiente y se compone de un conjunto de dispositivos que surgen del análisis de Iso esquemáticos de hardware.
- Los típicos dispositivos que están presentes en todos los chipsets (con las variantes de implementación de cada caso) son:
  1. Programación del chip de reloj
  2. GPIO. Igualmente debe consultarse el Chipset BIOS Writer Guide (NDA mediante) para tener los detalles.

8. Habilitaciones de dispositivos misceláneos

- Este ítem es plataforma dependiente y se compone de un conjunto de dispositivos que surgen del análisis de Iso esquemáticos de hardware.
- Los típicos dispositivos que están presentes en todos los chipsets (con las variantes de implementación de cada caso) son:
  1. Programación del chip de reloj
  2. GPIO. Igualmente debe consultarse el Chipset BIOS Writer Guide (NDA mediante) para tener los detalles.

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input/Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input /Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input /Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input /Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input /Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

9. Habilitaciones de Interrupciones

- Los procesadores de intel pueden manejar las interrupciones mediante cualquiera de los siguientes métodos o combinación de ellos:
  1. Programmable Interrupt Controller (PIC)
  2. Local Advanced Programmable Interrupt Controller (APIC)
  3. Input /Output Advanced Programmable Interrupt Controller (IOxAPIC)
  4. Messaged Signaled Interrupt (MSI)

Es hora de cargar un Sistema Operativo

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

¿Quien dará los pasos siguientes?

- La pregunta en esta instancia del desarrollo es: ¿Que recursos tenemos a la mano para usar?.
- Respuesta: Nuestro conocimiento del hardware de base, un compilador, un linker y alguna otra herramientas de desarrollo.
- Así comienza el desarrollo de cualquier computador.
- Incluso el de aquel simpático embedded que utilizaste hasta ahora.
- La pregunta que estás por hacerme es: ¿No están ya desarrollados?. ¿Para que necesitamos meternos en este problema?
- Mi respuesta es esta pregunta: ¿Te pusiste a pensar cual es la profesión del que diseñó la inicialización de tu PC o del embedded system que usaste en la carrera hasta ahora?. Pensá.....

Multiple choice. Necesario aprobar para seguir adelante

¿Cual es la skill de la gente que diseñó la inicialización de tu embedded?

- **a.** Bombero
- **b.** Odontólogo
- **c.** Fisioterapeuta
- **d.** Chef
- **e.** Licenciado en Ciencias de la Computación
- **f.** Vendedor de Seguros
- **g.** Periodista
- **h.** Abogado
- **i.** Actor

Multiple choice. Necesario aprobar para seguir adelante

¿Cual es la skill de la gente que diseñó la inicialización de tu embedded?

- **a.** Bombero
- **b.** Odontólogo
- **c.** Fisioterapeuta
- **d.** Chef
- **e.** Licenciado en Ciencias de la Computación
- **f.** Vendedor de Seguros
- **g.** Periodista
- **h.** Abogado
- **i.** Actor

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
  2. Cuando programamos el comportamiento es similar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
  2. Cuando programamos el comportamiento es similar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, fread, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la BIOS. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...).
     - En nombre de la productividad dejamos de pensar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la BIOS. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...).
     - En nombre de la productividad dejamos de pensar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la E/S. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la E/S. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...).

     En nombre de la productividad dejamos de pensar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la E/S. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.

Solo si aprobaste el Multiple choice.

La cuestión es dejar de pensar y actuar como usuarios

- **Diagnóstico**: Estamos acostumbrados a usar nuestra PC, aun para trabajar con un embebido...
- Esto nos induce a pensar como usuarios finales **siempre**.
  1. Cuando usamos aplicaciones en nuestra PC.
     - Instalación: ¿Alguna vez te compilaste una aplicación a partir de sus fuentes?.¿¡Compilar!? ¿Para que?. Gracias Wizards!!.
     - Buscamos el ícono llamado *Install*, *Setup*, o algo similar, Doble click al encontrarlo, y Botón Aceptar hasta las últimas consecuencias...
     - ¿Y cuando tenemos un problema?... botón de reset... :(
  2. Cuando programamos el comportamiento es similar.
     - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.). Esto es razonable.
     - Enviando requerimientos para acceder a la E/S. Esto también.
     - Usando librerías de código que nos facilitan la vida siempre que se pueda (no va a ser que usemos un código nuestro mas eficiente...). En nombre de la productividad dejamos de pensar.

Solo para profesionales de la computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...

Solo para profesionales de la computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...

Solo para profesionales de la computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...

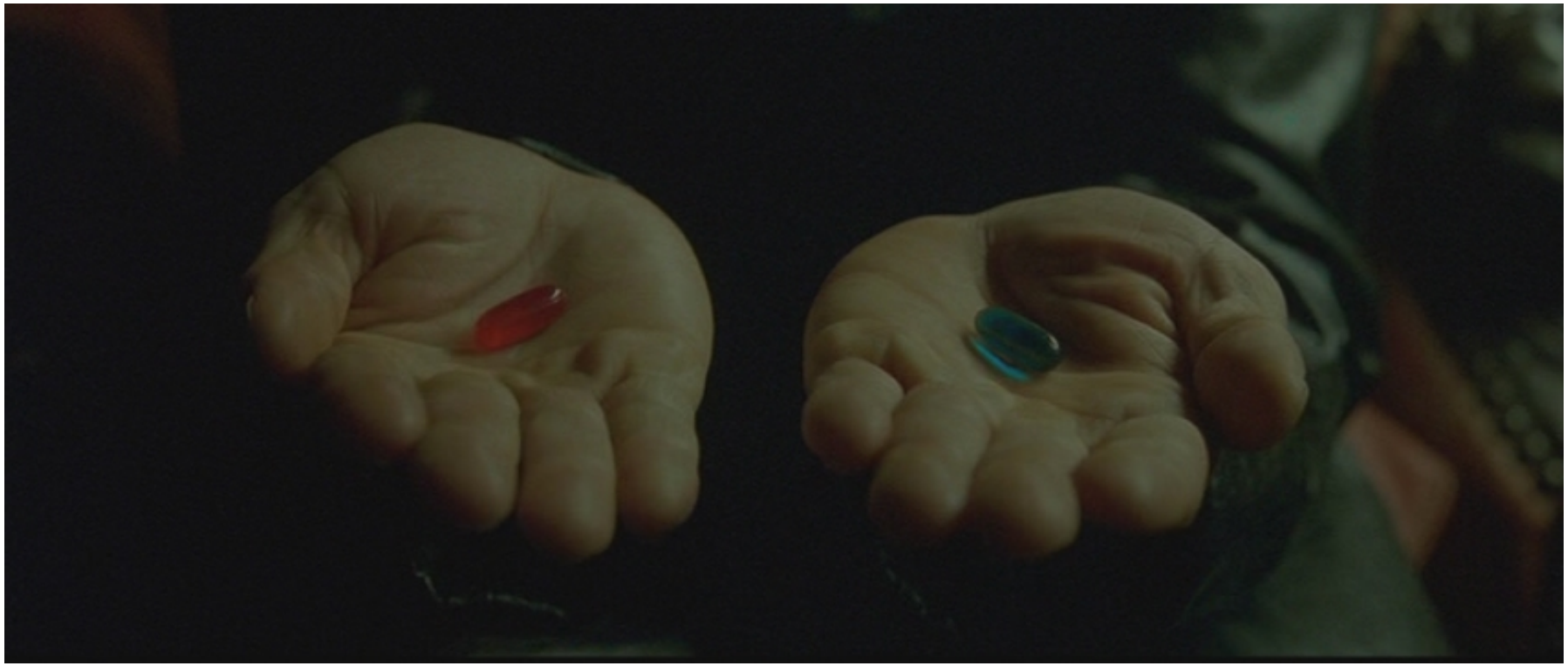
Solo para profesionales de la computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...

Solo para profesionales de la computación

- Lo dicho en el slide anterior no es válido para usuarios generales, como por ejemplo el resto de las opciones del múltiple choice anterior. Mas aún. Para este público el comportamiento descripto es el esperable.
- Pero Uds. son **otro público**, básicamente porque **eligieron serlo**. Por eso están aquí. ¿no?
- Así que a enterarse: Tu trabajo es y será siempre, **entender** como funcionan los sistemas electrónicos que te toque enfrentar, para poderlos diseñar, mejorar, o corregir. En este caso un computador, pero lo mismo vale para un transmisor de RF, o un sistema de control de lazo cerrado.
- Creeme: Entender cuesta. Pero hace la diferencia. Implica profundizar hasta dominar la tecnología. Algo que pesando como usuario no vas a conseguir...

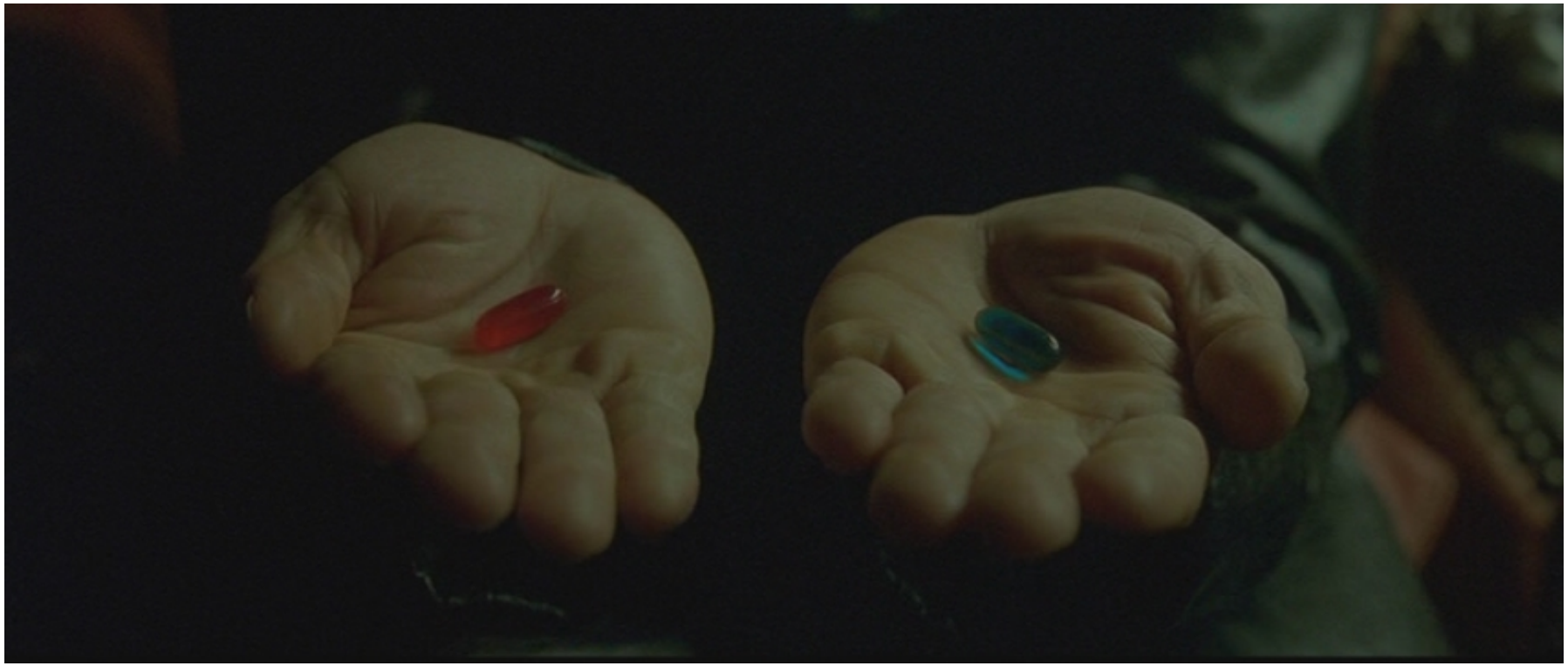
Es momento de revalidar la elección



The choice

O nos quedamos con el wizard que nos resuelve la vida sin tener que pensar... O nos decidimos a enfrentar las cosas como son realmente, y entenderlas, aprendiendo, si es necesario, a hacer todo desde cero y a pulmón.

Es momento de revalidar la elección



The choice

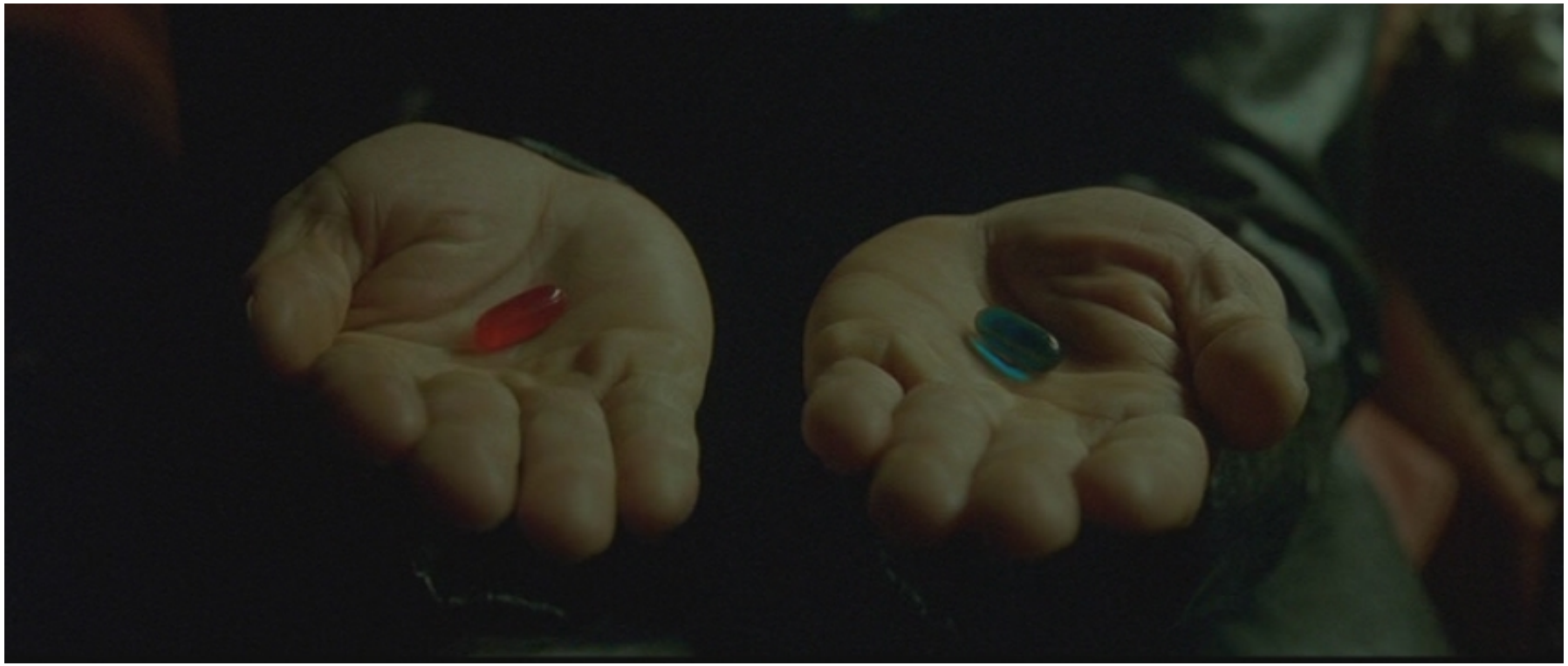
O nos quedamos con el wizard que nos resuelve la vida sin tener que pensar... O nos decidimos a enfrentar las cosas como son realmente, y entenderlas, aprendiendo, si es necesario, a hacer todo desde cero y a pulmón.

Retomando nuestro problema

Arrancamos el procesador y estamos en modo real

Para iniciar su operación en modo real no se requiere mas que inicializar un sistema de interrupciones con las reglas de modo real (para un 8086 diseñado en 1978), con los vectores de interrupción apuntando a código diseñado para operar en Modo Real, y tener disponibles las correspondientes funciones para cada handler de interrupción.
Si nos pensamos quedar en este pequeño microclima de confort (desperdiciando el 99,999 % de los recursos del procesador :-/), se necesita un mínimo kernel que administre la ejecución de los programas que compongan en rango de aplicaciones, ejecutándolas una a la vez (leíste bien... Primero una , y recién cuando finaliza, podés ejecutar otra aplicación).

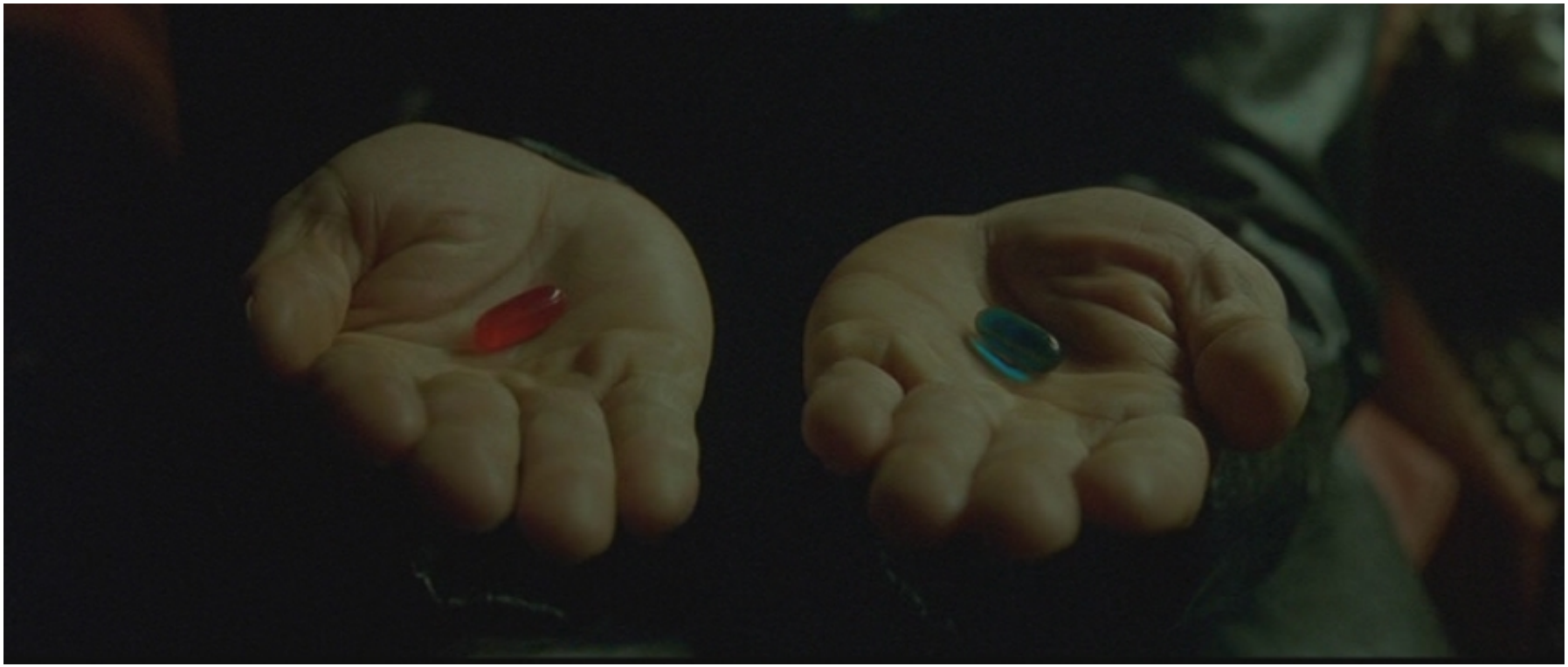
Otra vez...



The choice

La píldora azul nos deja en este nanoclima confortable. Fin de la presentación.
La píldora roja nos lleva al mundo real, donde utilizaremos TODOS los recursos del procesador para construir un Sistema Operativo...aunque no nos guste lo que vamos a encontrar.

Otra vez...



The choice

La píldora azul nos deja en este nanoclima confortable. Fin de la presentación.
La píldora roja nos lleva al mundo real, donde utilizaremos TODOS los recursos del procesador para construir un Sistema Operativo...aunque no nos guste lo que vamos a encontrar.

FAQ's

Q: ¿Vamos a desarrollar un sistema operativo???

A: La construcción de un Sistema Operativo es a veces una tarea gigantesca, como por ejemplo Linux, pero en ocasiones puede requerir un número mucho menor de rutinas que, aunque de muy bajo nivel, provean un conjunto de recursos base suficientes para administrar un sistema de menor tamaño, como un embeeded system.

Q: ¿Vamos a desarrollar un sistema operativo???

A: La construcción de un Sistema Operativo es a veces una tarea gigantesca, como por ejemplo Linux, pero en ocasiones puede requerir un número mucho menor de rutinas que, aunque de muy bajo nivel, provean un conjunto de recursos base suficientes para administrar un sistema de menor tamaño, como un embeeded system.

Q: ¿Para que?¿O acaso un embedded system no puede hostear Linux?

A: Por supuesto. Abundan implementaciones de Linux para Embedded Systems. Las embedded PC basadas en procesadores Atom, tambien pueden aceptar un Linux cualquiera. Aunque no todos los sistemas tienen un procesador con recursos de hardware suficiente para soportar Linux (por ahora... en cinco años hablamos otra vez).

Q: ¿Para que?¿O acaso un embedded system no puede hostear Linux?

A: Por supuesto. Abundan implementaciones de Linux para Embedded Systems. Las embedded PC basadas en procesadores Atom, tambien pueden aceptar un Linux cualquiera. Aunque no todos los sistemas tienen un procesador con recursos de hardware suficiente para soportar Linux (por ahora... en cinco años hablamos otra vez).

Q: ¿Para que necesitamos saber esto entonces? ¿No está todo hecho?

A: ¿Otra vez pensando como usuario? Además si un tal Linus Torvalds se hubiese quedado en esta pregunta hoy solo existiría Windows como alternativa para nuestra PC. (Vade retro!)

Q: ¿Para que necesitamos saber esto entonces? ¿No está todo hecho?

A: ¿Otra vez pensando como usuario? Además si un tal Linus Torvalds se hubiese quedado en esta pregunta hoy solo existiría Windows como alternativa para nuestra PC. (Vade retro!)

Ingresando a modo protegido

Requerimientos de los Sistemas Operativos Multitasking

- Área de memoria exclusiva para cada tarea para almacenar su código y sus datos. (Área Local).
- Área de memoria común a todas las aplicaciones, para que éstas puedan acceder a datos globales del sistema, o a código propio del Sistema Operativo de modo de permitir la comunicación entre las aplicaciones. (Área Global).
- Cada tarea podrá acceder únicamente a su Área Local y al Área Global, pero nunca podrá acceder al Área Local de otra tarea. De este modo el Sistema Operativo garantiza la integridad (PROTECCION ;)) del código y de los datos propios de cada tarea.
- Alta velocidad de procesamiento
- Gran capacidad de Direccionamiento de memoria

Requerimientos de los Sistemas Operativos Multitasking

- Amplio espacio de direccionamiento para memoria RAM
- Capacidad de Gestión de memoria de cada tarea por el método de Memoria Virtual
- Capacidad de implementar Multitarea de manera rápida y segura.
- En cada momento la CPU ejecuta una tarea de la lista que mantiene, poniendo a su disposición todos los recursos de hardware de la máquina, incluyendo la cantidad de memoria requerida por la aplicación.

Finalmente... ¿Que es Modo Protegido?

- Es el conjunto de recursos de hardware (Dije Hardware. O sea Electrónica.) y sus reglas de funcionamiento que se requieren para darle sustento a un sistema operativo multitasking satisfaciendo los requerimientos anteriores.
- Su dominio permite entender como funcionan las cosas en un sistema real que puede ser un súper servidor, o un embedded.Y en definitiva es nuestro trabajo.
- Es decir,... es tomar la píldora roja.

¿Como encarar la tarea?

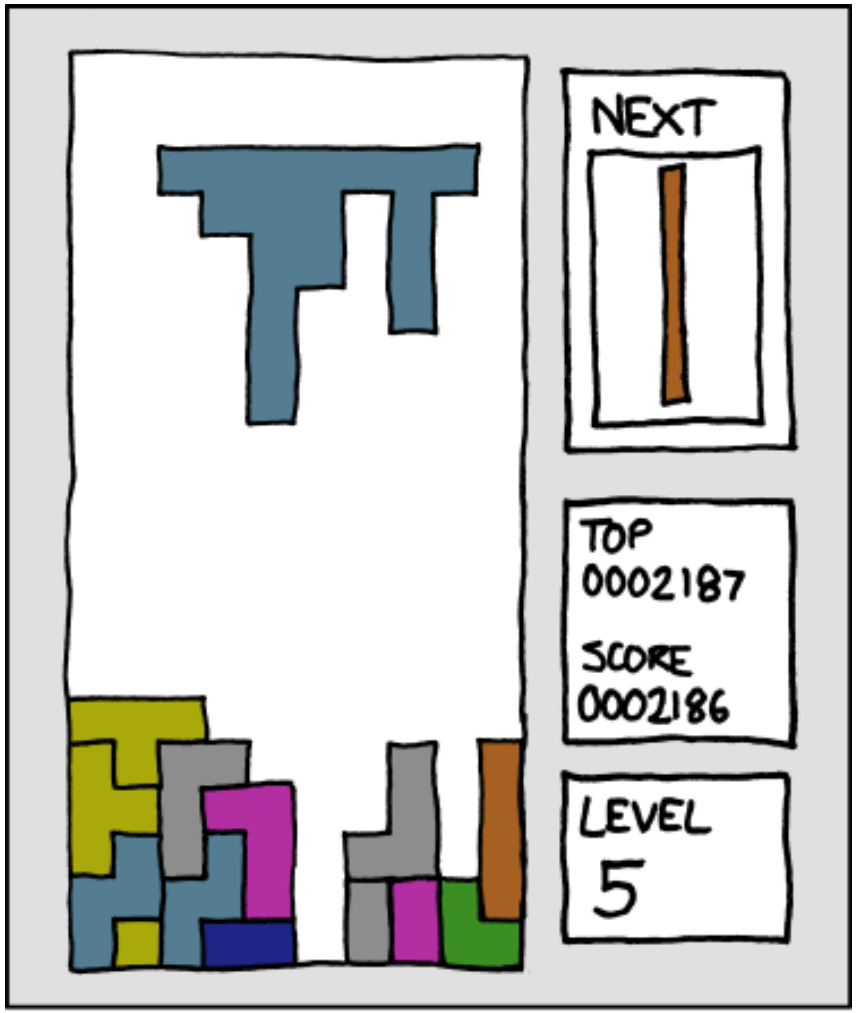
Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.

¿Como encarar la tarea?

Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.

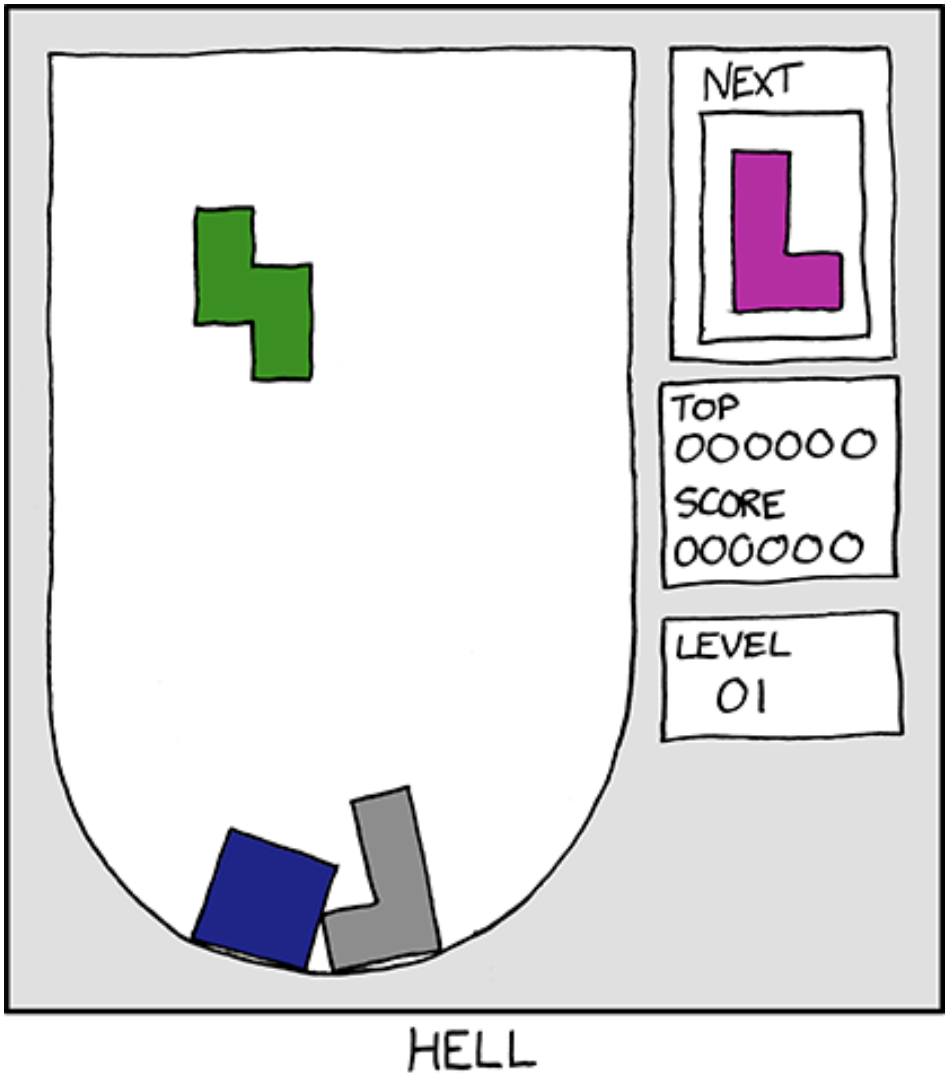
No vamos a mentir. No es fácil

Programación a nivel de aplicación...



En ocasiones vas a experimentar algo como...

Programación en Modo Protegido desde cero...



Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea `A20`) que simplemente no existía en el 8086. El resultado es `0xFFFF:0xFFFF = 0x0FFEF`.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea **A20**) que simplemente no existía en el 8086. El resultado es **0xFFFF:0xFFFF = 0x0FFEF**.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea `A20`) que simplemente no existía en el 8086. El resultado es `0xFFFF:0xFFFF = 0x0FFEF`.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea **`A20`**) que simplemente no existía en el 8086. El resultado es **`0xFFFF:0xFFFF = 0x0FFEF`**.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea **`A20`**) que simplemente no existía en el 8086. El resultado es **`0xFFFF:0xFFFF = 0x0FFEF`**.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea **`A20`**) que simplemente no existía en el 8086. El resultado es **`0xFFFF:0xFFFF = 0x0FFEF`**.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

Problemática en Modo Real

- El 8086 restringido a 1Mbyte de Memoria, administrada por segmentos cuyo tamaño máximo era 64Kbytes, y soporte de rollover, generó prácticas de programación algo desprolijas
- Una direrección lógica 0xFFFF:0xFFFF, debido a la propiedad conocida como *wrap around* de los segmentos, daba como resultado la siguiente dirección física:

$$0xFFFF << 4 + 0xFFFF = 0x10FFEF$$

- El 1 mas significativo se descarta debido a que corresponde al bit 20 del bus de address (línea **`A20`**) que simplemente no existía en el 8086. El resultado es **`0xFFFF:0xFFFF = 0x0FFEF`**.
- Esta propiedad (sin entrar en valoraciones subjetivas acerca de su conveniencia) fue aprovechada por no pocos programadores.
- Los procesadores siguientes inician su operación en el denominado Modo Real a fin de ser compatibles.
- Por lo tanto se necesitó respetar este comportamiento. Pero todos podían acceder muy por encima del Mbyte.

El 80286 direccionaba 16Mbytes. ¿Entonces?

- En el ejemplo anterior A20 enviaba un 1 hacia el bus de address.
- Este no es el comportamiento esperado por un 8086. O dicho de otro modo por un programa que trabaja en Modo Real, que como dijimos especularon con que **A20** nunca sería '1'.
- Como al lanzar el primer procesador que tenía capacidad de direccionamiento por encima del Mbyte, el 80286, Intel no hizo nada al respecto, fueron los fabricantes de MotherBoards (IBM particularmente) quienes debieron lidiar con la situación

El 80286 direccionaba 16Mbytes. ¿Entonces?

- En el ejemplo anterior A20 enviaba un 1 hacia el bus de address.
- Este no es el comportamiento esperado por un 8086. O dicho de otro modo por un programa que trabaja en Modo Real, que como dijimos especularon con que `A20` nunca sería '1'.
- Como al lanzar el primer procesador que tenía capacidad de direccionamiento por encima del Mbyte, el 80286, Intel no hizo nada al respecto, fueron los fabricantes de MotherBoards (IBM particularmente) quienes debieron lidiar con la situación

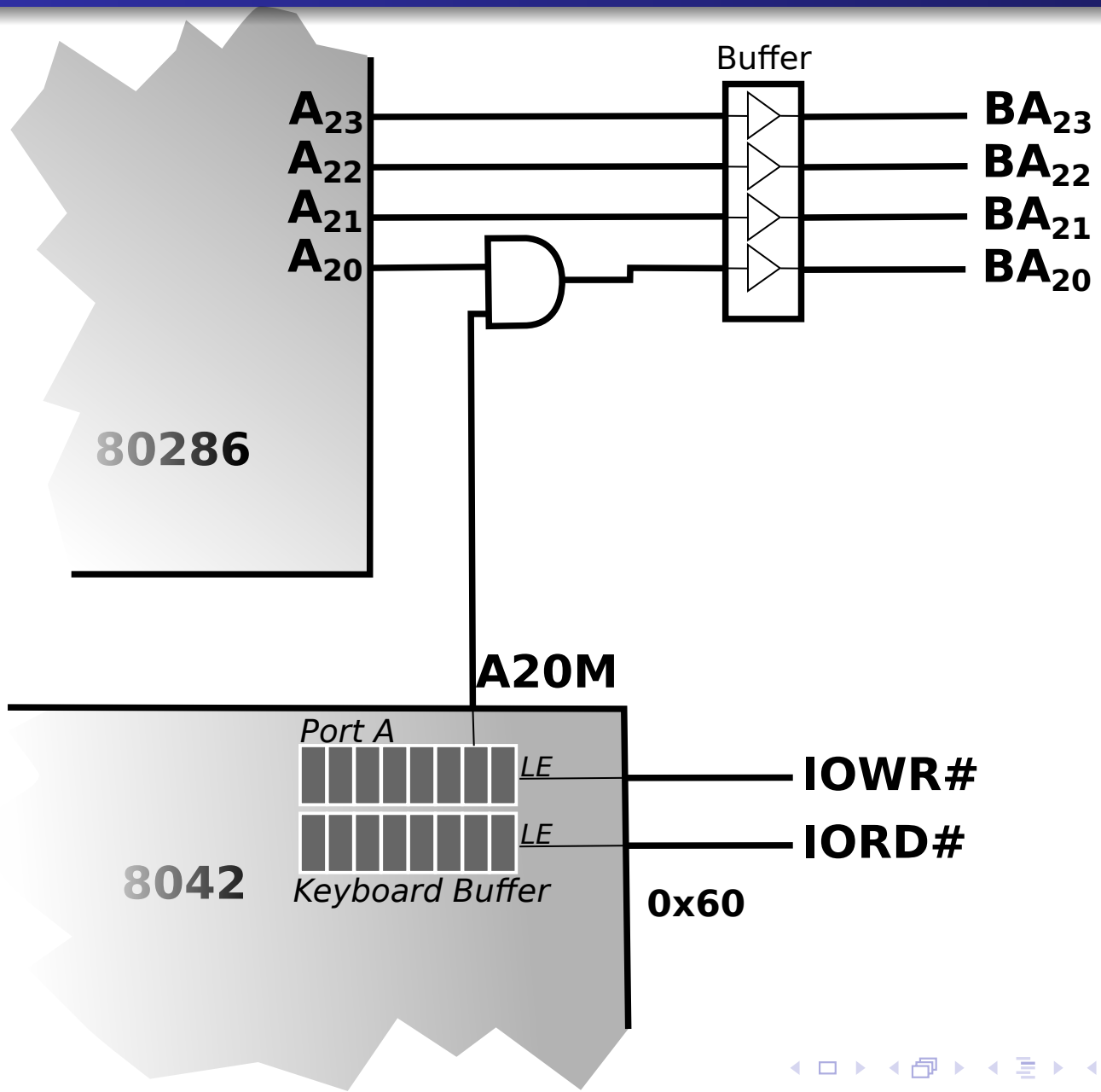
El 80286 direccionaba 16Mbytes. ¿Entonces?

- En el ejemplo anterior A20 enviaba un 1 hacia el bus de address.
- Este no es el comportamiento esperado por un 8086. O dicho de otro modo por un programa que trabaja en Modo Real, que como dijimos especularon con que **`A20`** nunca sería '1'.
- Como al lanzar el primer procesador que tenía capacidad de direccionamiento por encima del Mbyte, el 80286, Intel no hizo nada al respecto, fueron los fabricantes de MotherBoards (IBM particularmente) quienes debieron lidiar con la situación

El 80286 direccionaba 16Mbytes. ¿Entonces?

- En el ejemplo anterior A20 enviaba un 1 hacia el bus de address.
- Este no es el comportamiento esperado por un 8086. O dicho de otro modo por un programa que trabaja en Modo Real, que como dijimos especularon con que **`A20`** nunca sería '1'.
- Como al lanzar el primer procesador que tenía capacidad de direccionamiento por encima del Mbyte, el 80286, Intel no hizo nada al respecto, fueron los fabricantes de MotherBoards (IBM particularmente) quienes debieron lidiar con la situación

Hardware


Buffer
A23
A22
A21
A20
BA23
BA22
BA21
BA20
80286
A20M
Port A
LE
LE
IOWR#
IORD#
8042
Keyboard Buffer
0x60

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Funcionamiento

- Se activa o desactiva mediante un bit en el puerto de E/S 0x60.
- El port 0x60 corresponde al controlador 8042 cuya función principal es proveer los códigos de las teclas pulsadas en el teclado.
- Leer el port 0x60 permite obtener el código de la tecla recientemente pulsada.
- Escribir el port 0x60 permite establecer diversos comportamientos. En particular el bit de orden 1 es el encargado de habilitar o no la compuerta AND.
- Un '1'en ese bit permite a la compuerta AND colocar en su salida el estado de la línea A20 del procesador.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Soporte on chip

- Intel a partir del 80486 al incluir en el mismo chip, la CPU, el controlador cache y el cache level 1, incluye un terminal **A20M#**, que si se activa habilita el comportamiento compatible con el 8086.
- En el reset la línea A20 está habilitada (de otro modo no podría acceder a la dirección del reset vector en 0xFFFFFFF0, ya que el la línea A20 no podría estar en 1, lo que haría imposible sacar esa dirección al address bus.
- Para activar la función es necesario escribir '0' en el terminal del procesador **A20M#**.
- Siempre se hace en el bit 1 del port 0x60.
- Los fabricantes en el BIOS le activan un 0 en **A20M#** para que esté en modo 8086. Si el sistema operativo trabajará en modo real no debe hacer nada, pero si va a pasar a modo protegido deberá activar la línea **A20M#**.
- Desde la microarquitectura Haswell Intel ya no da soporte a esta funcionalidad.

Código

Nos proponemos agregar código de A20 y dejarlo listo para entrar a amodo protegido.
Luego un segundo round de Linker script