Cika Electrónica

Tutorial: CTU-011
Título: **Introducción a Holtek desde MCS-51**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	14/02/07	
1	16/02/07	directiva OFFSET

El presente tutorial se orienta a introducir a Holtek a los ingenieros y developers familiarizados con MCS-51, de una forma práctica y concisa. Para un análisis de la arquitectura y descripción del uso de los periféricos más comunes, se sugiere el tutorial CTU-005.

Índice de contenido

Comparación de Arquitecturas

La arquitectura de Holtek es en cierto modo similar a la de MCS-51. Haciendo algunas salvedades, podemos asumir que Holtek es una especie de subset de la arquitectura MCS-51, lo cual haremos para este análisis, resaltando aquellos factores arquitectónicos que los acercan o diferencian:

En ambos:

- El tradicional acumulador A se ve como un SFR más (ACC), permitiendo que una gran cantidad de operaciones sólo realizables sobre la memoria puedan efectuarse en el acumulador
- La arquitectura permite interrupciones vectorizadas, aunque en realidad se trata de offsets fijos. Una interrupción ocasiona un salto a una posición fija en memoria de instrucciones, dependiente de la causa de la interrupción. El contenido del PC se guarda en el stack (hardware en Holtek, RAM interna en MCS-51) para permitir su recuperación al ejecutar una instrucción de retorno, que restablece el estado de las interrupciones. No se salva el contexto de la CPU, el programa debe salvar los registros necesarios en alguna posición de memoria (o el stack en

MCS-51). Es posible prescindir de salvar el registro de estado (STATUS en Holtek, PSW en MCS-51) si el interrupt handler realiza funciones clásicas.

- MCS-51 dispone de 4 bancos de 8 registros en RAM interna, seleccionables mediante un SFR. El espacio de memoria interna de 0x00 a 0x80 es direccionable tanto directa como indirectamente (mediante dos de estos registros, R0 y R1), mientras que el espacio de 0x80 a 0xFF es solo direccionable de forma indirecta; los SFR ocupan el espacio de 0x80 a 0xFF en direccionamiento directo. Holtek dispone de una serie de SFRs y luego RAM interna, desde 0x00 hasta 0xFF, direccionable en forma directa o indirecta mediante un par de SFRs (uno es puntero y el acceso al otro realiza la indirección, dado que no hay opcodes para direccionamiento indirecto). Las operaciones lógicas se realizan de forma indistinta sobre cualquier posición de memoria interna.
- Es posible alojar tablas o constantes en memoria de programa, accediéndolas mediante una instrucción especial (MOVC en MCS-51, TABRDC/TABRDL en Holtek, mediante direccionamiento indexado (PC o DPTR) en MCS-51 e indirecto (TBLP) en Holtek.
- Ante una resta, el flag de Carry se utiliza como Borrow. MCS-51 presenta un borrow complementado, como muchos micros: $SUBB\,A,m \Rightarrow A = A - C - m$; Holtek presenta un borrow normal, como muchos DSP: $SBC\,A,m \Rightarrow A = A - m + \overline{C} = A + \overline{m} + C$.

Sólo Holtek:

- Emplea pipelining, todas las instrucciones demoran un ciclo de máquina (cuatro ciclos de clock) en ejecutarse, excepto aquellas que obligan a purgar el pipeline, como instrucciones de salto.
- Utiliza técnicas de VLIW al incorporar los datos en la instrucción, es decir, la memoria de instrucciones es de un ancho mayor, de modo de poder incluir constantes (valores inmediatos, direcciones de datos) en las instrucciones. Por ejemplo, muchos micros de esta serie tienen un ancho de palabra de instrucciones de 14-bits. En este espacio se distribuyen las más de 60 instrucciones disponibles, algunas de las cuales incluyen los 8 bits del dato correspondiente (direccionamiento inmediato) o su dirección en memoria de datos (direccionamiento directo). Esto limita la cantidad contigua de memoria de datos disponible a 256 bytes (2^8) .
- No permite cargar un dato inmediato en memoria sino que es necesario cargarlo primero en el registro A y luego cargar éste en memoria.
- Tiene el program counter accesible en la memoria de datos, es un SFR más. Dado que la memoria de datos es de 8-bits, los bits (14, 15 ó 16) del PC están partidos en PCL (8-bits) y PCH (6, 7 ú 8 bits). El registro PCH no es accesible directamente, por lo que las instrucciones que modifiquen el flujo mediante cálculo y operación sobre PCL (por oposición al uso de CALL o JMP) están limitadas a un segmento de 256 words de programa. Las operaciones de CALL y JMP operan sobre la totalidad de la memoria de programa sin restricciones ni bank switching.
- No posee instrucciones de manejo de stack.
- Afecta el STATUS ante operaciones aritméticas y lógicas solamente, mover un dato al acumulador no altera el flag de Zero.
- No posee direccionamiento directo de bit y álgebra booleana en el sentido del MCS-51, sí permite acceder a un bit dentro de un byte.
- No dispone de memoria externa
- Dispone del acumulador como operando obligado en toda operación aritmética o lógica. El resultado puede depositarse en la memoria si ésta es el otro operando. Esto atenúa el inconveniente de disponer de sólo un registro en la CPU, dado que puede hacerse que una determinada operación no altere el acumulador, guardando el resultado en la memoria de datos.

Comparación

MCS-51

Presenta direccionamiento lineal hasta 64K flash (programa y tablas, acceso indexado) y 64K RAM (datos, acceso indirecto). Existe un espacio de 256 bytes interno con acceso directo/indirecto o indirecto, según la dirección.

Se trata de un CISC con arquitectura Harvard modificada, stack en RAM interna. Diferentes CPUs de diversos fabricantes aceptan clocks de hasta más de 40 MHz, internamente divididos por doce o no, lo que permite una ejecución de más de 40 MIPS pico, que suelen disminuir bastante en operación sostenida, dependiendo del algoritmo, dado que en los micros 'single-cycle' por lo general la mayoría de las operaciones se ejecutan en varios ciclos de clock.

Como periféricos, encontramos una vasta variedad de dispositivos, dependiendo de la inventiva del fabricante. Por lo general todos disponen de una UART o dos y dos o tres timers de 8 ó 16-bits.

Holtek series “Cost Effective”, “I/O” y “A/D”

Presenta direccionamiento lineal hasta 8K OTP/MTP y 224 bytes RAM; el modelo que incorpora una cantidad mayor de RAM emplea bank switching.

Se trata de un RISC con arquitectura Harvard modificada, stack en hardware de 2 a 16 niveles, según el modelo. Su operación es bastante similar a PICmicro de Microchip, pero sin el uso de bank switching en flash ni I/O y practicamente nulo en RAM. La CPU acepta un clock de hasta 8 MHz, internamente dividido por cuatro, lo que permite una ejecución a 2 MIPS pico, que no disminuyen demasiado en operación sostenida, dado que la gran mayoría de las operaciones se ejecutan en un ciclo de clock.

Como periféricos, encontramos un timer de 8-bits en casi todos los modelos; y ninguno, uno o dos timers de 16-bits en las series “I/O” y “A/D”. Los timers tienen capacidad de controlar un buzzer mediante dos salidas complementarias con 50% de ciclo de trabajo. Los modelos de la serie “A/D” incorporan un conversor AD de 9 ó 10-bits de 4 ú 8 canales, y los modelos más avanzados agregan interfaz I^2C y generador de PWM. El watchdog timer posee un oscilador independiente.

Paralelo de hardware

Dada la gran variedad de micros MCS-51, no haremos una comparativa de características similares de hardware en esta oportunidad. Además, la mayoría de los MCS-51 del mercado apuntan a una gama más alta. Es la intención de este tutorial facilitar al migración en los casos en que un micro de gama más baja como Holtek sea una alternativa más interesante.

Paralelo de software

A continuación haremos un paralelo de software.

Las comparaciones de uso de memoria de programa serán uno a uno. Si bien MCS-51 emplea bytes y Holtek words, comparamos ambos procesadores por su capacidad en unidades elementales (1K byte vs. 1K word). Lo hacemos así dado que la capacidad de memoria de programa de ambos está especificada de esta forma, y nos permite realizar una comparación directa. La cantidad de ciclos está basada en los ciclos de máquina de un MCS-51 standard, en el cual un ciclo de máquina son doce períodos de clock. Diferentes implementaciones tendrán diferente timing, y por lo general los micros single-cycle necesitan más de un ciclo de clock en muchas de las operaciones, por lo que una comparación contra un determinado micro deberá realizarse tomando en cuenta su timing particular. Respecto al assembler de Holtek, el mismo determina mediante las declaraciones qué nombres corresponden a variables y cuáles se trata de constantes, generando automáticamente direccionamiento directo o inmediato según corresponda. Ante una ambigüedad, existe la directiva OFFSET. En cuanto a los espacios de memoria de datos, dada la arquitectura de Holtek, limitaremos MCS-51 al manejo de memoria interna, que generalmente es más rápido y requiere instrucciones más cortas.

Interrupción

Evaluamos el tiempo necesario en atender una interrupción, salvar contexto, y retomar ejecución normal del programa. En Holtek no disponemos de instrucciones para el stack, pero las instrucciones de *mov*e no afectan los flags.

Con uso del acumulador y flags:

```
push psw
push A
...
pop A
pop psw
reti
```

10 ciclos + overhead

Sin uso del acumulador ni flags (mover un pin en un port).

```
clr P2.1
setb P2.0
...
reti
```

4 ciclos + overhead

Con uso del acumulador y flags:

```
mov somewhere,A              ; salva A
mov A,STATUS
```

```
        mov somewhere2,A              ; salva STATUS
        ...
        mov A,somewhere2
        mov STATUS,A
        mov A,somewhere
        reti
```

8 ciclos + overhead = 10 ciclos
Sin uso del acumulador ni flags (mover un pin en un port).

```
        clr PA.1
        set PA.0
        ...
        reti
```

4 ciclos + overhead = 6 ciclos

Bifurcación (branch) según flag

Este es el típico caso de modificación del flujo de programa debido a circunstancias especificadas en un flag. Dada la carencia de direccionamiento relativo, Holtek lo resuelve mediante una instrucción de salteo (skip) y un salto. Generalmente, la bifurcación debe pensarse al revés

```
        jnb bit,l1            ; si el bit está dentro del espacio de memoria direccionable como bits
        ...
        jnb sfr.bit,l1        ; si se trata de un SFR direccionable por bits (ACC, B, ports, etc)
        ...
        ; caso contrario, debe llevarse a un SFR direccionable por bits (MOV, MOVX)
l1:
```

2 ciclos, 3 bytes

```
        snz var.bit           ; saltea instrucción si el bit 'bit' de la variable 'var' es 1
        jmp l1                ; caso contrario salta a 'l1'
        ...
l1:
```

2/3 ciclos, 2 words

Salteo (skip) según flag

Esta situación se produce generalmente cuando debemos alterar el estado de un pin según nos indica un flag, pero no podemos darnos el lujo de producir glitches (modificarlo primero asumiendo un valor y después corregirlo si el flag indica otra cosa). Consideramos solamente el tiempo empleado en la decisión. En MCS-51 asumimos que el flag está en un SFR o posición de memoria direccionable por bits

```
        jnb var.bit,l0
        ; acción si 1
        sjmp l1
l0:     ; acción si 0
l1:
```

4 ciclos, 5 bytes

```
        sz var.bit
        ; acción si 1
        snz var.bit
        ; acción si 0
```

3 ciclos, 2 words

En un caso más simple, cuando sí nos podemos permitir el lujo de un glitch o simplemente modificamos un flag:

```
        ; acción si 0
        jnb var.bit,l1
        ; acción si 1
l1:
```

2 ciclos, 3 bytes

```
        ; acción si 0
        sz var.bit
        ; acción si 1
```

2 ciclos, 1 word

Modificar flag

Alteramos el estado (seteamos o reseteamos) de un bit en una variable

```
        setb var.bit            ; si el flag está en el espacio de memoria/SFR direccionable como bits

        orl var,(1<<bit)        ; caso contrario. (1<<bit) puede no ser aceptado por el assembler
```
1(2) ciclos, 2(3) bytes

```
        set var.bit
```
1 ciclo, 1 word

Típico loop

Consideramos solamente el overhead que introduce lo necesario para producir el loop, es decir, decremento del contador y salto al inicio del loop.

```
l1:
        ...
        djnz R2,l1              ; decrementa R2 y si éste no es 0 vuelve al loop
        ; o bien
        djnz var,l1             ; decrementa 'var' y si ésta no es 0 vuelve al loop
```
2 ciclos, 2 (3) bytes

```
l1:
        ...
        sdz var                 ; decrementa 'var' y si ésta es 0, saltea la siguiente instrucción
        jmp l1                  ; vuelve al loop
```
3 ciclos, 2 words

Sumar valor a variable 8 bits

Sumamos un determinado valor constante a una variable cualquiera; el resultado debe quedar en la variable en cuestión.

```
        mov A,#value
        add A,var
        mov var,A               ; resultado de la suma en var
```
3 ciclos, 6 bytes

```
        mov A,value
        addm A,var              ; resultado de la suma en var
```
2 ciclos, 2 words

Lo mismo, pero el resultado queda en el acumulador

```
        mov A,#value
        add A,var
```
2 ciclos, 4 bytes

```
        mov A,value
        add A,var               ; resultado de la suma en A
```
2 ciclos, 2 words

Sumar una variable a otra, 8 bits

Sumamos dos variables entre sí; el resultado debe quedar en la última variable direccionada.

```
        mov A,var1
        add A,var2
        mov var2,A              ; resultado de la suma en var2
```
3 ciclos, 6 bytes

```
        mov A,var1
        addm A,var2             ; resultado de la suma en var2
```
2 ciclos, 2 words

Memory move, inmediato 8 bits

Colocamos un valor constante en una posición de memoria.

```
        mov var,#value
```
1 ciclo, 2 bytes

```
        mov A,value
        mov var,A
```
2 ciclos, 2 words

Memory move, directo 8 bits

Colocamos en una variable el valor contenido en otra.

```
        mov var2,var1
```
2 ciclos, 3 bytes

```
        mov A,var1
        mov var2,A
```
2 ciclos, 2 words

Cálculo en 16 bits: K = next_T - advance – COMP_K

Realizamos un pequeño cálculo no muy complejo, en el cual obtenemos el valor de una variable de proceso que depende de otras dos variables y una constante de calibración. Las variables son todas de 16-bits, y la constante de calibración es de 8 bits, para mostrar un cálculo combinado.

```
        clr C
        mov A,next_Tl
        subb A,advancel             ; next_T – advance (LSB)
        mov Kl,A
        mov A,next_Th
        subb A,advanceh             ; MSB
        mov Kh,A
        mov A,Kl
        clr C
        subb A,COMP_K               ; - COMP_K (8 bits)
        mov Kl,A
        jnc done                    ; carry ?
        dec Kh                      ; sí, Kh = Kh – 1
done:
```
14 ciclos, 24 bytes

```
        mov A,next_Tl
        sub A,advancel              ; next_T – advance (LSB)
        mov Kl,A
        mov A,next_Th
        sbc A,advanceh              ; MSB
        mov Kh,A
        mov A,Kl
        sub A,COMP_K                ; - COMP_K (8 bits)
        mov Kl,A
        snz C                       ; borrow ?
        dec Kh                      ; sí, Kh = Kh – 1
done:
```
11 ciclos, 11 words

Cálculo simple en 32 bits: sumar dos variables

Sumamos dos variables en 32-bits, el resultado lo guardamos en una de ellas.

```
        mov A,var1ll
        add A,var2ll                ; suma LSB
        mov var2ll,A
        mov A,var1lh
        addc A,var2lh               ; acarrea
        mov var2lh,A
        mov A,var1hl
        addc A,var2hl
        mov var2hl,A
        mov A,var1hh
        addc A,var2hh               ; MSB
        mov var2hh,A
```
12 ciclos, 24 bytes

```
        mov A,var1ll
        addm A,var2ll               ; suma LSB
        mov a,var1lh
        adcm A,var2lh               ; acarrea
```

```
        mov a,var1hl
        adcm A,var2hl
        mov a,var1hh
        adcm A,var2hh                ; MSB
```
8 ciclos, 8 words

Máquina de estados

Analizamos a continuación un dispatcher del tipo comúnmente utilizado en máquinas de estados, en el cual el flujo de programa es redirigido según el valor de una variable, es decir, el estado de la máquina. En el caso de Holtek, la función y la tabla de saltos deberán estar dentro de un mismo segmento de 256 words. En el caso de MCS-51, existen 128 estados posibles como máximo.

```
        mov DPTR,#STATETABLE
        mov A,STATE             ; estados posibles: 0,1,2,3,4,etc;
        add A,ACC               ; x2, AJMP es una instrucción de 2 bytes
        jmp @A+DPTR             ; resultado de A+DPTR en PC
STATETABLE:
        ajmp State1
        ajmp State2
```
8 ciclos, 2 bytes por estado

```
        mov A,STATE             ; estados posibles: 0,1,2,3,4,etc
        addm A,PCL              ; resultado de la suma en PCL
        jmp State1
        jmp State2
```
5 ciclos, 1 word por estado

Table read, tabla en RAM

Tomamos un valor de una tabla según nos indica una variable que oficia de índice dentro de la misma. La tabla se encuentra en RAM. En Holtek el tamaño máximo de la tabla es de 224 bytes, en MCS-51, utilizando memoria interna, dejando espacio para el stack y algunos registros, tal vez podamos extendernos algunos bytes más.

```
        mov A,#TABLA            ; tabla
        add A,OFFSET            ; + offset
        mov R0,A
        mov A,@R0               ; en memoria interna (el destino puede ser cualquier variable o SFR)
```
4 ciclos, 8 bytes, resultado en A

```
        mov A,OFFSET TABLA      ; tabla
        add A,OFFST             ; + offset
        mov MP,A                ; indirecto
        mov A,IAR
```
4 ciclos, 4 words, resultado en A

Table read, tabla en memoria de programa

Tomamos un valor de una tabla según nos indica una variable que oficia de índice dentro de la misma. La tabla se encuentra en ROM. En el caso de Holtek, la tabla puede ocupar un espacio dentro del segmento de 256 bytes en memoria de programa en el que se está ejecutando el código, o en el último del mapa de memoria, las tablas no deben pasar un segmento de 256 unidades de memoria de programa.

```
        mov DPTR,#TABLA
        mov A,OFFSET
        movc A,@A+DPTR
        ...
TABLA:
```
5 ciclos, 6 bytes+tabla, resultado en A, si se desea en *var* debe agregarse un mov adicional

```
        mov A,OFFST
        add A,TABLA
        mov TBLP,A
        tabrdl var              ; tabla en última página de 256 bytes en memoria de programa
        ...
tablas  .section at LASTPAGE 'code'
TABLA   dc dato1, dato2,...
```
5 ciclos, 4 words+tabla, resultado en *var* (si se desea en A, *var=ACC*)

Direccionamiento indirecto en RAM

Tomamos un valor de un buffer en RAM e incrementamos el puntero

```
        mov R0,#TABLA           ; tabla en memoria interna
        mov A,@R0               ; el destino puede ser cualquier variable o SFR
        inc R0
```
3 ciclos, 4 bytes

```
        mov A,tabla
        mov MP,A
        mov A,IAR
        inc MP                  ; incrementa puntero
```
4 ciclos, 4 words

Conversión binario a BCD

Finalmente, evaluaremos una subrutina muy común, como la empleada para presentar resultados en un display de 7-segmentos o LCD alfanumérico, es decir, la tradicional conversión de un byte a BCD.

```
BINBCD: mov R2,#8               ; COUNTER=8
        mov RESULT,#0           ; RESULT=0
L1:     mov A,SOURCE
        rlc A
        mov SOURCE,A            ; rota SOURCE a la izquierda (MSB a Cy)
        mov A,RESULT            ; no afecta Cy
        adc A,RESULT            ; suma A + RESULT + Cy, resultado en A (A=2 x RESULT + Cy)
        da A                    ; ajusta A para que sea BCD válido
        mov RESULT,A            ; resultado en RESULT
        djnz R2,L1              ; decrementa R2 y si éste no es 0 vuelve al loop
        ret                     ; resultado en RESULT
```
76 ciclos, 19 bytes

```
BINBCD: mov A,8
        mov COUNTER,A           ; COUNTER=8
        clr RESULT              ; RESULT=0
L1:     rlc SOURCE              ; rota SOURCE a la izquierda (MSB a Cy)
        mov A,RESULT            ; no afecta Cy
        adc A,RESULT            ; suma A + RESULT + Cy, resultado en A (A=2 x RESULT + Cy)
        daa RESULT              ; ajusta A para que sea BCD válido, resultado en RESULT
        sdz COUNTER             ; decrementa 'COUNTER' y si éste es 0, saltea la siguiente instrucción
        jmp L1                  ; vuelve al loop
        ret                     ; resultado en RESULT
```
60 ciclos, 10 words