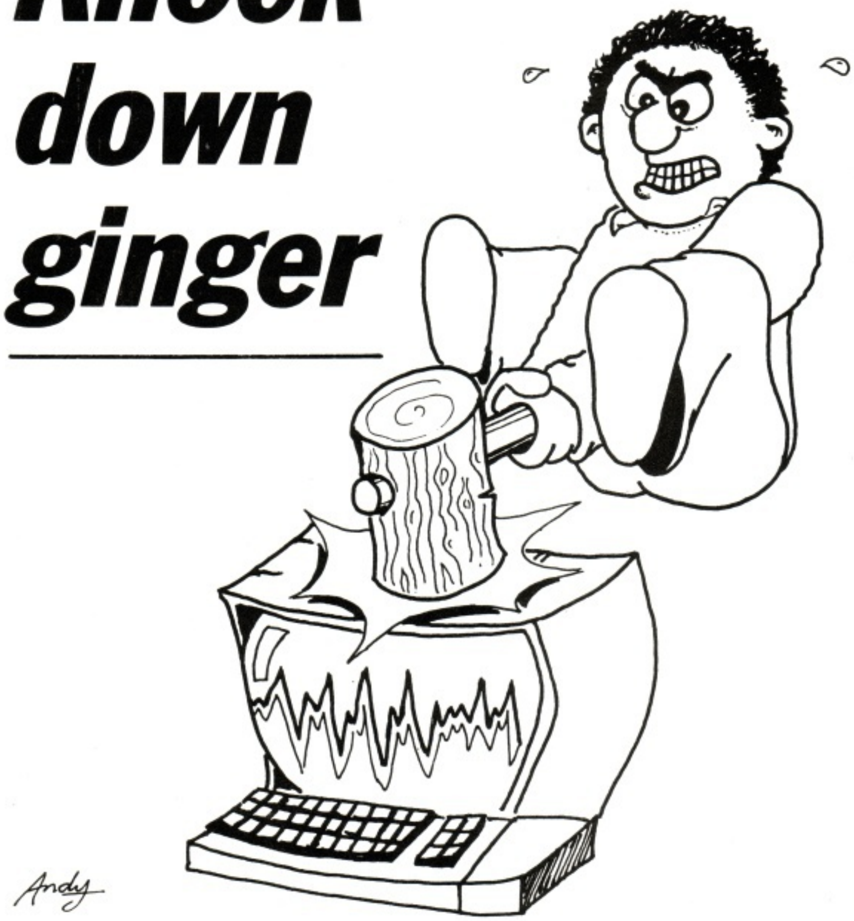
Knock down ginger



Auntie John gets in the mood with some interrupt-driven machine code sixteen bar blues

FESTIVE greetings. Before you crumple up this month's column and use it to gift wrap a turkey, why not try reading it for a change? It's not as painful as it looks, and who knows, you might even enjoy it. With the holiday season in full swing, everybody's thoughts are turning to questions like:

- How does Father Christmas get down the chimney without leaving soot all over the living room floor?
- What colour socks will Auntie Ethel get me?
- How do you use interrupts on the CPC?
- Why won't Fiona go out with me?

I can answer at least one of these questions, and if you have any suggestions for the others you should get in touch with me via the Editor. Thank you.

Interrupts

Once upon a time my friend Colin was watching a video. Colin has a low boredom threshold, so he soon got tired of this and started watching the television set instead. Now Colin has also been to a university, so he isn't stupid – he switched the TV on and started to watch a film.

```
;
; The interrupt driven machine code sixteen bar blues.
;
kl_init_event equ &bcef
kl_add_ticker equ &bce9
sound_chip equ &bd34
org &4000           ;This is where it's at, man.
ld hl,tickblock+6
ld b,3+128          ;Event class.
ld c,0              ;Rom selection.
ld de,eventcode     ;Address of code to be executed.
call kl_init_event
;
ld hl,tickblock
ld de,100           ;Initial delay time before first event.
ld bc,10            ;Time between interrupts.
call kl_add_ticker
;
call reset_tune
;
ret                 ;Return to Basic.
tickblock ds 13
eventcode
;
; The code that follows is executed every
; time an interrupt is generated.
;
ld hl,(count)
ld a,(hl)
ld e,a
inc hl
ld a,(hl)           ;Get tune data from memory,
ld d,a              ;and put it in DE pair.
inc hl
ld (count),hl
or e
cp 0                ;Check for end-of-data marker.
jp z,reset_tune
;
ld l,e              ;Copy data into HL pair.
ld h,d
;
srl d               ;Divide DE by 4, see text for reason.
rr e
srl d
rr e
;
ld a,0              ;Data for channel one.
ld c,l              ;That's an ell not a one.
call sound_chip
ld a,1              ;That's a one not an ell.
ld c,h
call sound_chip
;
ld a,2              ;Data for channel two.
ld c,e
```

Listing I: An interrupt driven tune

He had just settled down with a cup of hot coffee and some unformatted chocolate digestives when he heard the doorbell ring. He got up and opened the front door, but there was no one there. Deciding that some of the friendly neighbourhood kids must have been playing a trick on him, he closed the door, rewound the video and sat down again.

After only a few seconds the doorbell rang again. Colin lept up, ran to the door and flung it open, but again there was no one there. He shouted for a bit at some puzzled-looking kids playing on the other side of the road, then went back indoors.

He had only just sat down after rewinding the video when the bell went again. Just as he was about to jump up he noticed that one of the actors in the film he was watching also got up and answered the front door. The doorbell Colin had heard was the one in the film.

Now you might think the only reason I told you that story was to embarrass Colin. Nothing could be further from the truth, I simply wanted to introduce you to interrupts.

Colin had been quietly watching a film when he had been interrupted. He dealt with the interrupt, and then returned to watching the film.

Computer interrupts operate in a very similar manner. A program is in the act of being executed when an interrupt occurs. The computer processes the interrupt – that is, control passes to the interrupt handling routine – and then returns to running the original program.

Just how the interrupts are "serviced" depends on the microprocessor used, the hardware of the system, and how much sleep the electronic engineer who designed the computer got the night before. Even the humble Z80 inside Arnold has three different interrupt modes, not to mention non-maskable interrupts. Each would need several hundred words of explanation.

Fortunately those nice chaps who built Arnold have supplied us with some routines in the firmware to make things almost easy. Notice that I said almost – interrupts are difficult at the best of times, never mind around Christmas.

Non-stop music

As an example, I'm going to show you how to use interrupts to make a tune play constantly in the background. By "background" I mean other programs – Basic, machine code, even your word processor rom – will run at the same time and independently of the tune. As this is incredibly annoying, I'll also show you how to stop the tune from playing.

A routine is needed that will play a note every time it is called – if you called the routine 10 times in a row, it would play the first 10 notes of the tune. The shorter the time between calling the routine, the faster the tune will appear to play.

Imagine a monkey playing a barrel organ. Every time he turns the handle a note is produced. The interrupt is the bloke who feeds the monkey. The more often he gives the monkey a peanut, the faster the monkey turns the handle and the faster the tune gets. Or something like that.

Such a routine has been incorporated into Listing I, following the label *eventcode*. Although the program looks long and involved, don't let it put you off – it can be separated into two smaller programs; one to play the tune and one to set-up the interrupts.

The data for the tune is stored in byte form, along with a special end-of-data marker. When this is reached the program must go back to the beginning and start again. Notice also that two bytes – given the label *count* – are set aside to allow the routine to remember how far along the tune data it has reached. The two bytes point to the address where the next piece of data is to be found. Each time the data is read, this pointer is incremented (has one added to it) so it points to the next piece of data. If the end of the data is reached, the pointer is given the address of the first data byte, back at the start of the list. Thus every time this routine is called, one piece of data is read from a list that determines the note played by the sound chip.

The firmware calls used to create the sound were discussed in the December issue, so against my bank manager's advice – I get paid by the word – I'm not going to go over the details again. But bear in mind that the sound chip is being accessed directly to avoid any nasty internal disputes.

The reason why DE is divided by four at one point is to produce a slightly choral effect when played. You should also bear in mind that I have no musical ability whatsoever.

Tick beep tock beep

Now for the interrupts themselves. The firmware fully supports interrupts and allows very flexible use to be made of them. This means that describing every possible option would take a lot of time and space, so for a fuller description you are advised to sneak a look at the Firmware Guide (Soft 968, available from the User Club).

From the simplest point of view, there can be any number of individual "events" executed after a specified time period. The operating system (OS) automatically takes care of everything; you just tell it where the code you want executed is and how often you want it done.

The OS keeps a special list in mind, and every 1/50th of a second it consults the list and checks to see if any events need doing. If so, it gets all

```
call sound_chip
ld a,3
ld c,d
call sound_chip
;
ld a,8              ;Allow envelopes on both
ld c,16             ;channels one and two.
call sound_chip
ld a,9
ld c,16
call sound_chip
;
ld a,11             ;Define the period
ld c,100            ;of the envelope.
call sound_chip
ld a,12
ld c,10
call sound_chip
;
ld a,13             ;Choose the envelope.
ld c,1              ;That's a one not an ell.
call sound_chip
;
ld a,7              ;Set main register and make sound.
ld c,60
call sound_chip
;
ret                 ;Return from "eventcode" routine.
```

```
reset_tune
;
; Set "count" to point to start of the tune data.
;
ld hl,tunedata
ld (count),hl
jp eventcode
tunedata
;
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 956,1073,1276,1073,1276,1423,1276,1517
dw 1432,1432,716,716,804,956,804,716,1432
dw 1432,716,716,804,956,804,716,1432,1432
dw 716,716,804,956,804,716,716,358,804,402
dw 1073,536,1204,602
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 1911,1911,1517,1517,1432,1432,1351,1276
dw 956,1073,1276,1073,1276,1423,1276,1517
dw 1276,1276,638,638,716,851,716,638,1276
dw 1276,1012,1012,956,956,902,851,1432,1432
dw 1136,1136,1073,1073,1012,956,716,804,956
dw 804,956,1073,956,1136
dw 0 ;End-of-data marker
count dw 0
end                 ;End of listing.
```

```
kl_del_ticker equ &bcec
ld hl,tickblock     ;The address of the ticker block.
call kl_del_ticker
ret
```

Listing II: Routine to remove the event from the list

the details from a small table in memory called the ticker block, and goes to the event.

The event is the piece of machine code you want to be executed – in our case the routine needed to play a note. The time period between events is specified in 1/50ths of a second. For example, you could have your routine serviced once a second to maintain a clock, or several times a second to play a tune, or every 10 minutes to remind you to have another cup of coffee.

Writing an event

To write an event, first you must create the ticker block, informing the firmware of the event's address, add the event detailed in the ticker block to the list of events, and RETurn to Basic. Then you add the event subroutine.

The ticker block is 13 bytes long and must lie in the range of addresses &3FFF to &BFFF. Creating this table and adding it to the list of events to be processed needs two separate firmware calls. The first, KL INIT EVENT, needs to be told where the event is and whether it is in rom or ram. In our case the value zero in *ld c,0* indicates ram.

The event class allows the events to be processed in different ways. The value 131 in *ld b,3+128* means "don't go out of your way or anything, it's not that important really".

The call to KL ADD TICKER sets up the frequency of the event, that is how many times a second it is executed. Both times are in 1/50ths of a second. A value of zero is treated specially, because the event is only processed once and then forgotten.

The first time specified – in the DE register pair – is the length of time you want the computer to wait around before making the first call to the event. The second value – in the BC pair – is the length of time between successive calls to the event. In Listing I the inital delay is two seconds, subsequent delays are only 1/5th of a second.

Finally, an optional call is KL DEL TICKER at &BCEC. If perchance you wanted to stop the wonderful music playing, you would load the HL register pair with the address of the ticker block and call KL TICKER (see Listing II).

Further ideas

As suggested earlier, another use for the interrupt routine is a real time clock. The event code would update the time every second and print it on the screen.

Unfortunately, whenever the CPC is accessing tape or disc all interrupts are halted, so the clock will run slow. This is due to the Special Theory of Relativity formulated by Albert Einstein, which states that moving clocks run slower and are shorter than stationary clocks.

A solution to the problem is to accelerate the entire universe, except your CPC, to a speed close to that of light every time the disc or tape is used. Another solution is not to buy clocks from stationers.

And so we reach the end of this month's exciting machine code instalment. I just have time to wish a very happy and peaceful Christmas to you all. Let's all try especially hard to be nice to one another.