Retrocomputing Stack Exchange is a question and answer site for vintage-computer hobbyists interested in restoring, preserving, and using the classic computer and gaming systems of yesteryear. It only takes a minute to sign up.

Sign up to join this community

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

Retrocomputing Beta

Did anyone ever use the extra set of registers on the Z80?

Asked 2 years, 2 months ago Active 5 months ago Viewed 9k times

34

11

The Z80 has the surprising feature of a second set of registers. I suppose these were intended to be used for rapid task switching or interrupt handling, though I think if I were programming a Z80 retrocomputer, I would be more likely to use them for fast access to global variables.

Such small snippets of Z80 code as I have seen, do not use them, but then, that's not surprising; they are something that would be expected to be only used in large programs. Back in the day, I was on 6502 machines, so I never had occasion to write anything nontrivial on the Z80.

Did anyone ever use that second register bank, either for its intended purpose, or just to get more registers within a single task?

z80

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

4 I could see the alternative registers being used for fast context switching, but I'm not sure how efficient they would act as fast global variables. To use them, you would have to swap register sets, (BC, DE, HL with their prime counterpart (AF could also be swapped with a different instruction)). Then you would have to preserve a copy of that data, perhaps onto the stack or into an index register, then swap the sets back. It would probably be quicker just to grab the variable directly from memory. – RichF Sep 30 '18 at 20:17

6 On TI-83 series calculators, the OS uses the commands for their intended purpose in the system interrupt code. – Misha Lavrov Sep 30 '18 at 23:55

4 I don't know for what purpose but the operating system of the **Sinclair ZX81** and the one of the **Applied Technologies Microbee 16** used these register banks. – Martin Rosenau Oct 1 '18 at 6:16

4 I used it all the time especially for graphics when I needed more register variables. But yes I used it sometimes also for comfortable ISR handling. IIRC some on screen monitors/debuggers used it to avoid stack usage (they where inside SCREEN VRAM and not changing rest of RAM). When I switch to x86 I was missing the extra register file a lot – Spektre Oct 1 '18 at 8:38

3 I *started* doing serious assembly on Z80 (long time ago), and used the alternative registers all the time for, basically, general purpose. Writing for 6502 after that was a huge pain :) – Zeus Oct 1 '18 at 15:35

16 Answers

Active Oldest Votes

34

The key to efficient programming on Z80 is to use registers as much as possible. I can easily believe that designers of Z80 intended the use of the alternative set of registers as an efficient way of context switching. However, the context switching does not tend to happen often enough to use the alternative set of registers only for that; the gains are simply not worth it most of the time. Hence, the good practice of Z80 programming is typically about using as many registers as possible and still use stack for saving registers during the interrupts.

Now, let me give you several ideas on how one would benefit from having two sets of equivalent registers. A typical pixel scrolling for 16 byte wide bitmap can look e.g. as follows:

```
 rl (hl) : dec l          ; repeated 16 times
```

What if one needs to scroll by 2 pixels at a time?

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

is the fastest way. OK, this is only using the second accumulator. Let us consider fast copying. The obvious

```
ld a,(hl) : ld (de),a : inc hl : inc de        ; 26 t-states
```

which is actually very slow. Unrolled

```
ldi             ; 16 t-states
```

is better and, in fact, is often acceptably fast. However, the fastest copiers are based on (semi-)unrolled code loading and saving the data via the stack, e.g. as follows:

```
ld sp,.. : pop af : pop bc : pop de : pop hl
exx : ex af,af' : pop af : pop bc : pop de : pop hl
ld sp,.. : push hl : push de : push bc : push af
exx : ex af,af' : push hl : push de : push bc : push af
; 10+10*4 + 4*2+10*4 + 10+11*4 + 4*2+11*4 = 204 t-states per 16 bytes
```

i.e. 12.75 t-states per byte. And note that this is not esoteric; variations of this idea were used in a huge number of commercial games on ZX Spectrum.

Much non-trivial code, e.g. fast polygon fillers or texture mappers are only possible with decent speed if one uses both sets of registers simultaneously.

answered Sep 30 '18 at 21:26
introspec
3,139 1 13 17

3 I bet they did not, because if they did, the command set would have been very different. But the same can be said about pretty much every popular architechture: once it becomes popular, people find innovative ways to exploit it. And given how the software library for, say, ZX Spectrum, is absolutely dominated by game titles, I'd never call graphics fiddling esoteric... – introspec Oct 1 '18 at 13:30

8 Interrupts don't have to happen *often* for there to be value in rapidly switching to the interrupt context - it's the *latency* of response that motivates

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

2 By my count, the code you give to scroll by one pixel takes 19 cycles per byte, and the code to scroll by two takes 42. Using `ld a,(hl) / rla / ld (hl),a / inc l` would take 22, and if there were a way to adapt that to shift two bits at an extra cost of 16 cycles or fewer that could be a win, but the fact that swapping flags also swaps `A` would mean the cheapest approach I can see would cost 20 extra cycles. – supercat Oct 2 '18 at 18:29

1 @TobySpeight, my answer is likely to be coloured by my experience of coding small home computers, mostly ZX Spectrum-compatibles. Some of the most basic ones did not have any sources of interrupts; ZX Spectrum compatibles have a ~50Hz frame interrupt that does not really require any kind of response from the coder. I would actually love to learn about common Z80-based architectures where the response latency mattered, because I do not know a single example of this situation. – introspec Oct 2 '18 at 21:53

3 @cat, I do fair amount of Z80 coding even nowadays and I find traditional form of assembly code (one command per line) incredibly diluted. Thus, I use assembler that allows multiple commands per line (colon is a command separator, just like you guessed). I find that putting related groups of commands together onto a single line increases the readability, because you use the screen space better and can also group commands by their intended action. I recognise that some people find it off-putting. I hope you can recognise that I find the more traditional format just as off-putting. – introspec Oct 2 '18 at 21:59

22

> The Z80 has the surprising feature of a second set of registers. I suppose these were intended to be used for rapid task switching or interrupt handling,

Indeed they were intended for fast interrupt reaction. In a simple, general way, this saved the time to push the main process's registers onto the stack and restore them again. They even went so far to spend one of the very few available single-byte opcodes to get the absolute minimum execution time - as the Z80 Technical Manual states on p.26:

> OP code `08H` allows the programmer to switch between the two pairs of accumulator flag registers while `D9H` allows the programmer to switch between the duplicate set of six general purpose registers. These OP codes are only one byte in length to absolutely minimize the time necessary to perform the exchange so that the duplicate banks can be used to effect very fast interrupt response times.

`EX` and `EXX` only take 4 T-cycles, while even just pushing a simple 16-bit register would take 11 cycles, plus another 15 to load it again. 8 T-cycles instead of 25 or more cycles is a considerably faster reaction, isn't it?

That's also why there are two `EX*` instructions, as very simple routines may only (use and) need to preserve the flags and `A`.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

After all, the second set can not only be used for some kind of fast 'stack' but also be prepared for a certain I/O operation. Think maybe of a serial interface receiving at high speed. Loading things like the memory pointer where received data is to be placed, the number of bytes to receive and so on, does take quite some time (16 T-Cycles for a 16 Bit pointer, 13 for a byte value) - and they need to be stored later on as well.

If these values are placed in the second register set before the high speed interrupt-driven routine gets active, no loads and stores are to be executed. Interrupt service time gets reduced to the absolute minimum, not only causing less interruption of the main process but also working up to higher speeds.

The Z80 design was quite focused on a more flexible, configurable and faster interrupt handling than the 8080.

> though I think if I were programming a Z80 retro computer, I would be more likely to use them for fast access to global variables.

I can't see much gain here. Sure, 6 additional bytes or 3 pointers, but at the same time, you can't access the other ones. So there are not many cases where the secondary register set is helpful - besides interrupts and 'dead end' subroutines.

> Such small snippets of Z80 code as I have seen, do not use them, but then, that's not surprising; they are something that would be expected to be only used in large programs.

Well, it's exactly the region where they are useful - to speed up small functions.

> Back in the day, I was on 6502 machines, so I never had occasion to write anything non-trivial on the Z80.

Did both, and while they need different approaches, the result is usually quite similar.

> Did anyone ever use that second register bank, either for its intended purpose, or just to get more registers within a single task?

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

edited Feb 17 at 10:42
Toby Speight
727 7 18

answered Sep 30 '18 at 20:45
Raffzahn
120k 15 313 481

1 So within a single task, the most likely place to use them would be in leaf subroutines, so you can have the use of a full set of registers without having to spend cycles saving and restoring those used by the rest of the program. Okay, that makes sense. – rwallace Sep 30 '18 at 20:51

1 @rwallace yes, except there's till the issue of parameter passing. – Raffzahn Sep 30 '18 at 21:20

3 I miss writing code like this, with execution times and cycles in mind. So much code written today ignores what was such a fundamental concept then. – coteyr Oct 1 '18 at 7:35

Can you elaborate what you mean with "dead end" routines (last sentence)? – tofro Oct 1 '18 at 7:53

1 By "dead-end" subroutines, do you mean what compiler-writers sometimes call "leaf-functions", i.e. functions which do not call anything else? – OmarL Oct 1 '18 at 8:07

15

For my Aquarius Micro-Expander I used the alternate register set to speed up text display.

The Mattel Aquarius has a character based screen that is physically 40 characters by 25 lines, but to stay out of the overscan area it only uses 38 characters by 24 lines. With such odd dimensions calculating the cursor address takes a lot of code, and combined with other overhead makes printing to the screen quite slow. I wanted to reduce the overhead as much as possible, and also have more flexible line width and positioning.

Here's my core code for putting a character on screen. HL' holds the current screen address and D' holds the cursor position on the line. Higher level routines test and manipulate these registers to wrap at line ends and scroll text in windows.

```
WinPrtChr:
       EXX
       LD    (HL),A       ; poke char into screen memory
       INC   HL           ; HL' = next screen address
       INC   D            ; D' = next screen x
       EXX
       RET
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
        DEC   D           ; D' = previous screen x
        EXX
        RET
```

Using alternate registers was much faster than storing the variables in RAM, and kept the normal registers free for other uses. The result was fewer memory accesses and faster operation in general. My code prints text into windows of arbitrary size about 20 times faster than the system routines.

Unfortunately the Aquarius system ROM also uses alternate registers when scanning the keyboard, so I had to wrap the keyboard routines with a pair of EXX's (to get back to the normal register set) and save the affected registers on the stack when getting keyboard input. However I didn't have to worry about interrupts (where the alternate register set is often used for fast context saving) because the Aquarius doesn't have any!

answered Oct 1 '18 at 2:13
Bruce Abbott
2,995 6 15

14

The question asks if the alternate register set was "ever" used ... this answer is about a *modern* use of it.

The z88dk C compiler, `sccz80` [1] uses a calling convention where the registers B', C', D', E', H' and L' are used to hold the first 48-bit floating point argument to a function, or a floating point return value. Benchmark results suggest that this is a very good strategy, as the results are similar in performance to the 32-bit floating point used by most other Z80 C compilers.

[1] - actually, one of the two C compilers that are part of the z88dk toolchain, the other being a patched version of sdcc that uses 32-bit floats with a different calling convention

answered Sep 30 '18 at 22:30
Jules
11.5k 2 32 57

As you say one of the float libraries uses the exx set to hold the main floating point accumulator which is 48-bit in BCDEHL. Two floating point

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

The fastest integer math (multiply and divide) also involves the exx set. You'll see it used in 32-,64-,72-bit math and combinations with lower bit sizes. In stdio, the exx set is used to hold tallies like number of characters output and counters for sending buffers one char at a time when a device can't do a block on its own. The main set is then used by the driver to communicate with the device. The exx set is used in more places than that but those are the important ones. It improves performance so much that it's not worthwhile to reserve for fast interrupt response. – aralbrec Oct 6 '18 at 6:33

12

A specific example is the chess program Sargon (written in 1978 or a bit earlier), which used them in a couple of leaf subroutines.

Search the assembler listing at http://smecers.appspot.com/govs/Oldies/Sargon.htm for the EXX instruction. (It's in routines XCHNG and NEXTAD)

The code is well documented, if anyone wants to explore the cost of alternative coding techniques.

Apart from Sargon, I'm pretty sure they were used in the code examples in some "how to write interrupt routines" books of the period, as a "quick" way to save all the registers, but I don't have any references or that.

answered Sep 30 '18 at 22:25
alephzero
4,405 2 14 26

It would be useful if you could briefly explain how and why the alternate registers were used. – cjs Aug 23 '19 at 20:34

11

> Did anyone ever use that second register bank, either for its intended purpose, or just to get more registers within a single task?

This being one occasion when a personal experience answer will do, `EXX` is ideal for the very specific task of multiplying a 16-bit 2d vector by a scalar, which makes it helpful for 2d vector graphics, and the projection part of 3d vector graphics.

Specifically:

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. ✕

- use `BC` and `BC'` for the working copy of the multiplicands; these will need shifting left on each iteration;
- use `HL` and `HL'` to accumulate the result; perform `ADD HL, BC` if carry is set after the `RRA`.

So the specific convenient observations are:

- you're juggling four 16-bit quantities, but they interact only in pairs;
- and using `EXX` lets you use the 16-bit arithmetic that's right there on the main instruction page.

answered Sep 30 '18 at 21:42
Tommy
26.1k 1 81 128

10

As usual, the Z80 Oral History provides some insight into the motivation behind the alternate registers and exchange instructions:

[Faggin]

> So I wanted to have a couple of index registers, more 16-bit operations, a better interrupt structure. The whole idea of doubling the number of registers. And I could exchange the register with an exchange instruction, the whole register set. That was an idea that I had used already on the Intel 4040. So that one could serve it up very fast if that was a necessity. And on and on.

[Shima]

> Thirdly, in order to support the highspeed task switching, in the beginning I asked to complete two sets of register files including the program counter. But it was too complicated for customer. Then we gave up on [the idea] of the two sets of general purpose registers.
>
> ...

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

(And while it may be obvious, it's worth pointing out that in the silicon implementation, the exchange instructions merely modify the state of the register file addressing logic.)

answered Oct 2 '18 at 8:35
Jeremy
431 2 9

3 I wonder how much it would have cost to have separate prime-select bits for BC, DE, and HL, and have 16 or 32 DD-prefix opcodes xor those bits along with AF/AF' (and perhaps the DE/HL selection bit) with bits from the second byte. That could have made the secondary registers a lot more useful. – supercat Oct 2 '18 at 18:43

@supercat yes, but not really for ISRs. They usually never care about the state of the CPU before the interrupt happens, so long as it can be restored. So `EXX` is exactly what they need. – OmarL May 15 '19 at 10:33

@Wilson: Almost any plausible interrupt that would need `exx` would also need `ex af,af` , so a two-byte instruction which could swap any combination of registers would be no slower than the combination of `ex af,af'` and `exx` , but replacing them with such a two-byte instruction would leave two more single-byte opcodes available for other purposes, an excellent choice of which would have been a form of RET that acknowledges interrupts, and enables them unless combined with a prefix flag, thus reducing the amount of instruction-decode logic needed in Z80 peripherals. – supercat May 15 '19 at 14:59

@Wilson: BTW, I've also sometimes wondered how much it would have cost to have a mode where the opcodes for `ld b,b` , `ld c,c` , etc. would act as prefixes that would cause most instructions that act upon `a` to use the indicated register instead, so the opcodes that on the 8080 would mean `ld e,e / add a,c` would instead be processed as `add e,c` ? Given that `AF` and `AF'` are kept in the same register file as all the other registers, I don't know that such an approach would require much extra circuitry, relative to the value it could provide. – supercat May 15 '19 at 15:03

Check out Ken Shirrif's analysis of the Z80. The thing with the ' registers is that they are not grouped by banks of registers normal / prime but are intertwined. Each register is itself a double value register using a smaller circuit to implement 2 values than if they had been full size separate registers. As someone said above, there is one selector input in the register telling if it is plain or prime value. – Patrick Schlüter Apr 27 at 11:55

8

Following the spirit of personal examples, here's one from when I was in high school: I, like everyone else, used a Ti 84 calculator for my Math classes. Said calculator used a Z80.

Out of a mixture of boredom and curiosity, I wrote a program that printed "WOZ IS ALWAYS THE ANSWER" any time the user hit the enter key. The program didn't need to be running for this to happen, and it could happen anywhere in the menu system.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

To do that, I used the chip's interrupts to check the input every 1/140th second. The thing is, each time I checked, I *had* to corrupt the A register. The input methods simply couldn't get around that. So, if I had just used A, the A register would be switching values for every other program run. That wouldn't work.

Instead, I switched into the shadow registers, did the input checking, switched back, and waited for the next call. It worked out pretty well, and did the job!

answered Oct 2 '18 at 5:21
Kyle
81 1

8

Back in the early 90s I used the alternate registers in 3 different ways:

1. For task switching in a dual task "operating system".
2. In a RAM-less control application, temperature sensors and relay control. (208 bits in total was enough).
3. In a floating point library. The latter was also done by several others (and better than my attempt), see for example http://www.andreadrian.de/oldcpu/Z80_number_cruncher.html

answered Oct 4 '18 at 20:55
Baard
313 1 6

4

A fragment from DES encryption/decryption routine, the so-called Initial Permutation step:

```
;-------------------------------------------------------------------;
; IPFORM - Initial Permutation. MAPS a 64 bit to 64 bit number      ;
;-------------------------------------------------------------------;
    POP     HL      ; Restore pointer to des data   +3
    LD      A,8     ; Loop Counter
IPFORM                      ;
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. ✕

```
    LD      A,(HL)          ; Alternate accumulator
    RLA                     ;
    RR      C               ; C will contain byte 5 of output
    RLA                     ;
    RR      E               ; E will contain byte 1 of output
    RLA                     ;
    RR      B               ; B will contain byte 6 of output
    RLA                     ;
    RR      D               ; D will contain byte 2 of output
    EXX                     ; SECONDARY SET
    RLA                     ;
    RR      C               ; C' will contain byte 7
    RLA                     ;
    RR      L               ; L' will contain byte 3
    RLA                     ;
    RR      B               ; B' will contain byte 8
    RLA                     ;
    RR      H               ; H' will contain byte 4
    EXX                     ; PRIMARY SET
    INC     HL              ; Point to next byte
    EX      AF,AF'          ; Swap accumulators back
    DEC     A               ; Decrement loop counter
    JP      NZ,IPFORML      ; Repeat loop 8 times
;-------------------------------; Store the result data
    PUSH    DE              ; Store bytes 1,2 (L0BLK+0) on stack    +4
    EXX                     ; SECONDARY SET
    PUSH    HL              ; Store bytes 3,4 (L0BLK+0) on stack    +5
    PUSH    BC              ;                                       +6
    EXX                     ; PRIMARY SET
```

And yes, DI/EI are there because the OS used EX* commands for fast context switch.

answered May 13 '19 at 10:27

tum_

141 1 4

3

Sorry to rain on any parades but, while it's been a long time since I did any serious Z-80 programming, I distinctly remember the alternate registers being one of the major broken features of the Z-80.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

- Speed up interrupt handling.
- Store alternate data in your main code.

The thing was you could not do both.

So if I was writing a device driver with an interrupt service routine, I could not use the alternate registers to save pushing and popping the main registers. I of course would need to avoid doing this in case the application programmer was using those registers.

And if I was writing application code, I could not use the alternate registers because an ISR could trample all over my data.

The overall result was that most programmers stayed away from the alternate register sets so that there code would run on more machines without crashing and burning. Any savings was swamped by the risk of shipping non-working code.

Safe code may be slower, but broken code does not run at all.

Of course if you have the luxury of controlling all aspects of the system and its coding, you were free to do as you please. I never had that luxury.

answered Oct 3 '18 at 0:42

Peter Camilleri
1,047 4 12

3 Since some of the tricks in the other answers depend on pointing `sp` into non-stack data structures, I think it's implicit that one would disable interrupts on the way into those tight inner loops. – hmakholm left over Monica Oct 3 '18 at 14:26

3 @HenningMakholm, funnily enough, some people came up with register use conventions that allowed them to keep interrupts enabled while using SP pointing at non-stack data. Of course, this was happening on micros, where programmers had the luxury of controlling every aspect of the system :) – introspec Oct 3 '18 at 18:03

1 I suppose that the luxury in question is a nice way of saying "Having to do all the work yourself." Does not sound so good when you put it that way! – Peter Camilleri Oct 4 '18 at 1:08

5 It's hardly fair to describe a feature as broken just because it doesn't fit your particular use case. No, you can't have your cake and eat it; the alternate registers will suit either of the purposes you described but not both together. – user3570736 Oct 4 '18 at 5:28

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Z80s sold for embedded systems far exceeded those sold for general purpose computers, making the your general purpose use a small side market, as far as overall CPU sales were concerned. – cjs Aug 23 '19 at 20:41

3

I remember well that the ZX Spectrum ROM software made good use of the alternate registers.

The floating-point calculator routines are invoked by a RST 28h instruction. Following that instruction are in-line function codes and literals. The calculator routine generally uses HL' as a pointer to those codes.

The calculator also uses the alternate registers to perform 32-bit additions and subtractions. In places, one 32-bit value is in HL/HL' and the other in DE/DE', allowing an ADD/EXX/ADC sequence to add the two.

Take a look through, the ROM source is available on the interweb. An add-up shows 117 uses of EXX and 13 uses of EX AF,AF'. I would imagine the ZX81 and even ZX80 may be similar, as the ZX Spectrum software was a progression of the ZX81's.

edited Feb 20 at 16:53

answered Feb 20 at 16:27
TonyM
1,257 1 7 17

2

I used it in float16 library: https://github.com/artyom-beilis/float16

It was very useful, for example af' stored the sign part of calculation and hl', de', bc' used extensively in division routine.

It is very useful to put there some temporary values that you need in different places and access them quickly afterwards

answered Feb 20 at 18:16
Artyom
213 6

On the system I'm most familiar with (the Amstrad CPC range of home computers) the alternate register set was reserved for use

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

called a routine stored in the system ROM it would exchange register sets, do its thing, then exchange sets again. And of course interrupts were handled by OS functions so would do the same.

answered Apr 27 at 0:42
DrHyde
121 1

2

Yes, certainly.

As pointed out, the other set is very useful for floating point arithmetic, so that is one area I'm using them.

I also found out a very handy use for them when writing a pipelined read ahead and write behind processes for the file system: I start one disk operation, switch bank, start the other one, switch bank and wait for the first one to complete. And so on.

Yet another use case for me is the RTL (Run Time Library) for interpreted and compiled Basic: while the RTL-functionality can most of the time be implemented using just one bank, part of the other bank is often handy for keeping the return address.

As for the AF' register, I have reserved it for the first level interrupt handling: just doing EX AF,AF' allows the interrupt routine to do for e.g. some math without messing up user processes flags.

So, in my Z180 based system, both banks (except the AF') are included in the process context (I'm running a home brew multiprocessing system).

answered Jun 17 at 11:36
OldTimer
41 1

1

I used them all the time.

I starting programming Z80 machine code when I was 14 or 15 on a Sinclair ZX Spectrum after being briefly introduced to it on a

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

On the Speccy if you were just writing a machine code routine you knew which registers the OS (BASIC interpreter) would need preserved, which may or may not have included any of the alt. registers, I can't remember any more. But you just saved them on the stack or in RAM and restored them before returning to BASIC.

If you were writing a game, you just took over the whole system and could use all registers as you pleased and never return to BASIC. In this case you didn't even have to worry about interrupts.

There was no multitasking, no real OS, no memory protection, no security model. In more advanced OSes like CP/M or possibly any system with disk drives as standard, I'm sure you would also know which registers you had to avoid or save and restore and things you had to have in a certain way to allow for system interrupts.

But to me they were just extra registers with some quirky properties, and in those days all registers had some quirky properties. If you needed more registers you'd swap them in and out taking into account that most of the registers swap at once. The accumulator only took the flags register with it, while all other registers swapped in one big group. It was just stuff you had to keep straight in your head when writing machine code. I assume compilers could do a good or bad job trying to juggle them as well. I never used a compiler in the 8-bit days.

answered Apr 27 at 5:32

hippietrail
3,197 10 29

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.